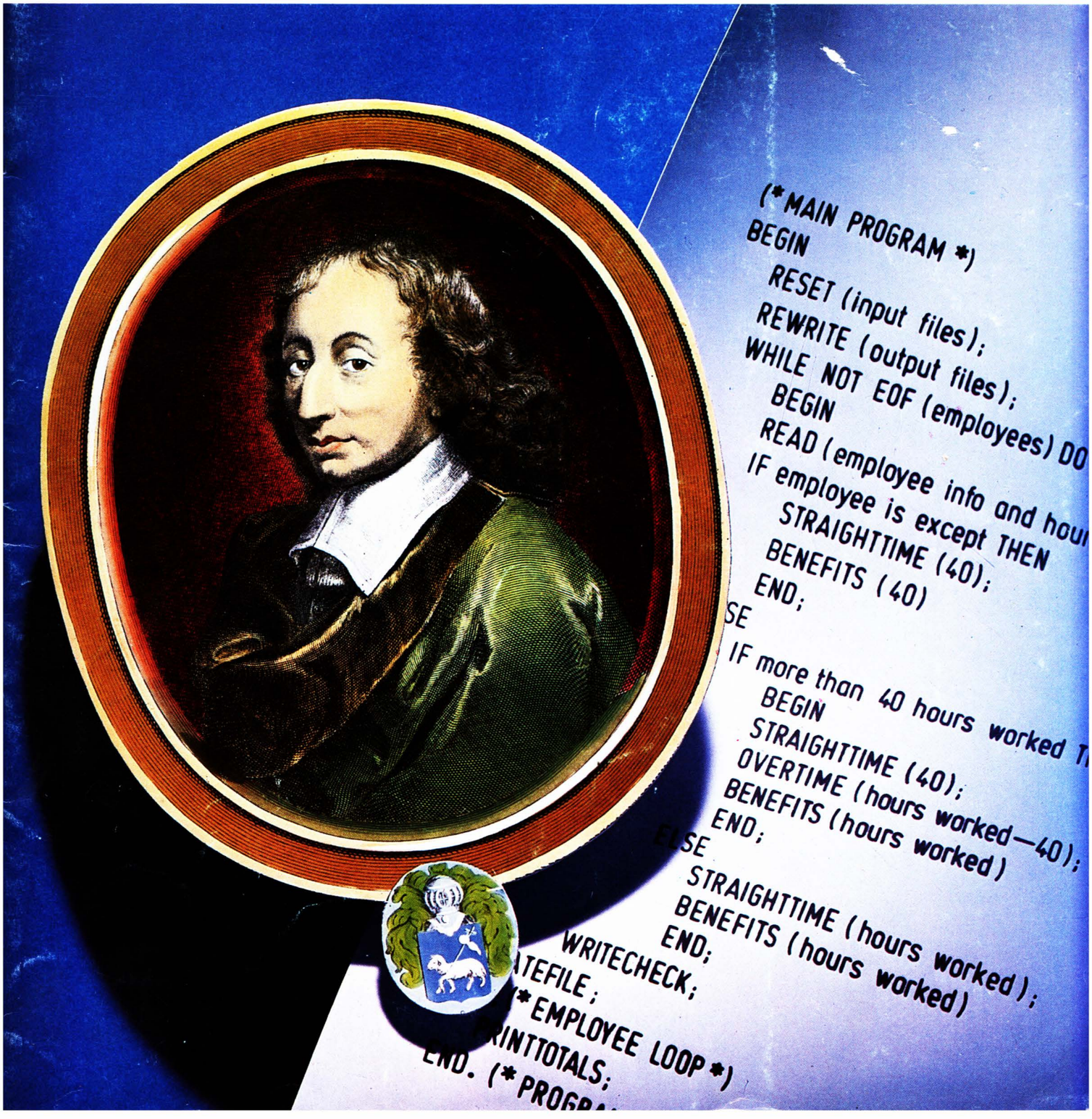
Elektronik
Sonderheft Nr. 46
Preis 18.50DM

PASCAL

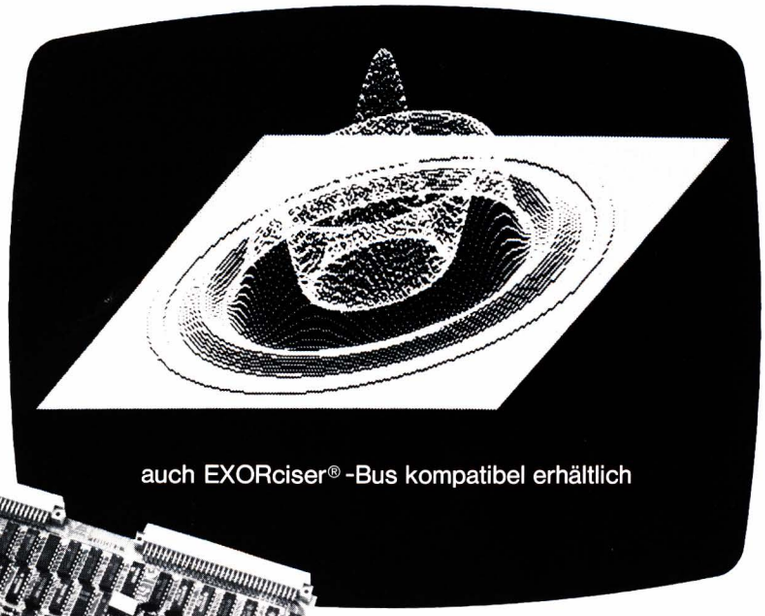
Grundlagen und Systeme



Computer-Grafik ?

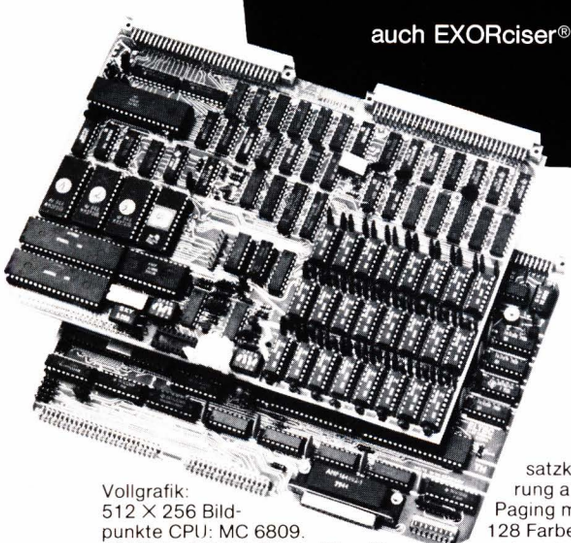
EUROCOM® II

leistungsfähig, ausbaufähig, preiswert



Abbildungen:

EUROCOM II; Controller mit Floppy und IEC-Bus Interface; hochwertige kapazitive Tastatur



Vollgrafik:
512 × 256 Bildpunkte CPU: MC 6809.
48 KByte RAM, 4 Steckplätze für EPROM. Schnittstellen: V 24, parallele I/O, Audio-Cassetten-Modem, Videoausgang. **DM 1.480,– +** MWST. (DM 1.672,40 incl. MWST.)
4 K Debug-Monitor in EPROM, Bedienung über Standard-ASCII-Tastatur.

Weitere Ausbaumöglichkeiten:
EPROM-Programmierzusatz • Mit EXORciser® -Bus erhältlich • Doppelte Auflösung 512 × 512 • RAM/Farberweiterung (96 KByte pro Zusatzkarte) • RAM-Erweiterung auf 240 KByte mit Paging möglich • Grafik bis zu 128 Farben/Graustufen • Fremdsynchronisation zur Einmischung des EUROCOM II - Videosignals in Videobilder • Low Cost Grafikdrucker. **Sonderanfertigungen:** EUROCOM II mit bis zu 1 MByte Bildspeicher zur Realtimeverarbeitung von Farbvideobildern.

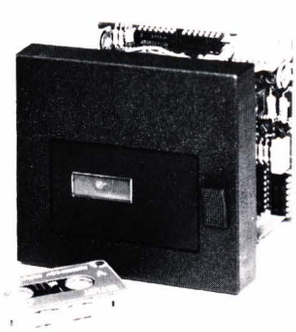
Software
für EUROCOM II mit Digital-Cassette.
Komfortabler Editor und Macro-Assembler
DM 198,– + MWST. (DM 223,74 incl. MWST.)

Basic mit Grafikfunktionen,
leistungsfähiges Basic 14 K, unterstützt voll die Grafik-Möglichkeiten des EUROCOM II. **198,– +** MWST. (DM 223,74 incl. MWST.).
Mehrseiten-Terminal-Programm,
mit Soft-scroll über 16 Seiten u. Grafik-Unterstützung, z.B. Tektronics 4010 Emulation, **DM 310,– +** MWST. (DM 350,30 incl. MWST.).
Für EUROCOM II mit Floppy:
Plattenbetriebssystem FLEX®, BASIC, Extended BASIC, BASIC Precompiler, Editor, Macro Assembler, Wordprozessor. Neu: **PASCAL.** Kundenspezifische Software kann erstellt werden.

Floppy-Controller

Floppy-Controller-Platine zu EUROCOM II Buskompatibel; für 5" und 8" Floppy-Drive; mit Floppy-Controller-Chip FD 1793; maximal 4 Drives, Double-Side; zusätzlich vollständiges normgerechtes IEC-Bus Interface.

DM 998,– + MWST. (DM 1.127,74 incl. MWST.) Mit Direct-Memory-Access und Double Density
DM 1.498,– + MWST. (DM 1.692,74 incl. MWST.)



Mini-DCR

Preiswerter Massenspeicher.

Philips Mini-Digital-Cassetten-Recorder mit Interface zum direkten Anschluß an EUROCOM II.

80 KByte formatiert, 6000 Baud Aufzeichnungsrate. Treiber-Programm für 2 Drives auf EUROCOM II unterstützt Cassettenbetrieb.
DM 490,– + MWST. (DM 553,70 incl. MWST.)

KHS 8–A–0

Kostenlosen Katalog anfordern.

Eltec Elektronik GmbH, Neubrunnenstr. 10, 6500 Mainz
Postfach 1847, Telefon 06131/26411, Telex 04187273

ELTEC

Vorwort

Die Entwicklung von Programmiersprachen begann vor mehr als 25 Jahren mit FORTRAN und ALGOL. Ziel war und ist die Schaffung eines Formalismus, der gestattet, Abläufe von Datenverarbeitungsprozessen, sog. Algorithmen, auf einem höheren Niveau der Abstraktion auszudrücken, als dies mit den Instruktionen spezifischer Computer möglich ist. Die praktische Anwendung solcher Formalismen, heute „höhere Programmiersprache" genannt, hängt direkt von der Verfügbarkeit zugehöriger automatischer Übersetzer, von Compilern ab, welche die Programme in die entsprechenden Folgen von Computer-Instruktionen umwandeln. Solche Übersetzer sind selber komplizierte Programme, und waren bis vor kurzem Großrechnern vorbehalten.

Durch die rasante Entwicklung der Halbleitertechnik und das Entstehen von Mikroprozessoren mit beachtlicher Rechenleistung und Speichergröße sind Compiler auch auf Mikrorechnern einsetzbar geworden. Dadurch erklärt sich die enorme Zunahme des Interesses an höheren Programmiersprachen.

PASCAL wurde in den Jahren 1968 bis 1970 entwickelt, und zwar sowohl die Sprache selbst wie auch ein zugehöriger Compiler. Die Sprache entstand aus längeren Studien und nach intensivem Gebrauch von ALGOL. Von ALGOL 60 hat PASCAL die wesentliche Idee der Struktur von Anweisungen geerbt und diese auf Datenvereinbarungen ausgedehnt. Daraus ergab sich ein wesentlich breiteres Einsatzfeld im Vergleich zu ALGOL, das fast ausschließlich den Numerikern vorbehalten blieb. Durch die Entwicklung von portablen Compilern wurde auch eine rasche Verbreitung auf viele Rechnertypen möglich, nicht zuletzt deshalb, weil der PASCAL-Compiler in PASCAL selbst formuliert wurde, und damit die Vorteile der Programmierung auf eine höhere Abstraktionsebene in einem Gebiet bewies, das bis dahin ausschließlich Domäne der Assembler-Codierung geblieben war.

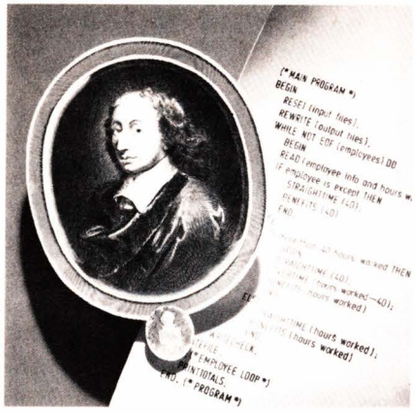
Es wäre aber irrig zu glauben, daß mit dem Erlernen einer höheren Sprache im allgemeinen, und von PASCAL im Speziellen die Schwierigkeiten des Programmierens beseitigt würden. Das Konzipieren von komplexen Systemen ist an sich eine anspruchsvolle geistige Tätigkeit. Sie fordert vom Konstrukteur Intelligenz, logisches und exaktes Denken, Erfindergeist und Erfahrung. Die Güte des verwendeten Werkzeugs kann aber dennoch den Erfolg entscheidend beeinflussen. Jedes Werkzeug bestimmt unseren Arbeitsstil und setzt Grenzen. Der Schritt vom Assembler-Code zur Programmiersprache ist für jeden Programmierer von einschneidender Bedeutung; er beginnt mit dem Erlernen einer Sprache und deren Konzepte, und er ebnet den Weg zu jahrelanger Sammlung von Erfahrung und zur Erarbeitung eines professionellen Programmierstils.

In den 10 Jahren seit dem ersten Einsatz von PASCAL ist die Forschung im Gebiet der Programmiertechnik und ihrer Werkzeuge nicht stillgestanden. Neue Erkenntnisse in der Programmierung von Systemen mit mehreren Prozessoren (sog. Multiprogrammierung) und für die geeignete Strukturierung von großen Systemen (Modularisierung) haben ihren Niederschlag in neueren Programmiersprachen gefunden. Sie werden in Zukunft noch mehr an Bedeutung gewinnen und PASCAL ergänzen.

Man hat im jungen Gebiet der Computertechnik nie ausgelernt. Was das Programmieren anbetrifft, ist aber sicher das Umdenken von Assembler-Codierung zur modernen Programmiersprache eine auf lange Sicht lohnende Investition unserer intellektuellen Arbeitskraft

N. Wirth

Prof. Niklaus Wirth



Unverhoffter Nachruhm

Als der große französische Denker und Gelehrte *Blaise Pascal* (1623–1662) eine der ersten funktionierenden mechanischen Rechenmaschinen konstruierte, ahnte er sicher noch nicht, daß man dergleichen rund 300 Jahre später ohne mechanische Teile im Westentaschenformat und für sehr wenig Geld bauen würde. Noch weniger aber ahnte er, daß einst sein Name – auch wieder im Zusammenhang mit der gebauten Rechenmaschine – als Name für eine höhere Programmiersprache gegen Ende des zwanzigsten Jahrhunderts erneut zu Ehren kommen würde. Unser Titelbild zeigt den Gelehrten nach einem Kupferstich von *Edelinck,* der aus dem Archiv des Deutschen Museums zur Verfügung gestellt wurde; daneben ein typisches PASCAL-Listing.

PASCAL

Nachdem in den letzten Jahren die Notwendigkeit zur Kostenreduzierung auf dem Gebiet der Mikroprozessor-Software immer größer wurde, hat sich gezeigt, daß ein geeignetes Mittel zur Lösung dieses Problems – zumindest aber zur Entschärfung der sich anbahnenden „Softwarekrise" – die Nutzung höherer Programmiersprachen auch für Mikroprozessoren ist. PASCAL hat sich aus bestimmten Gründen (die hier in diesem Heft erläutert werden) für diese Zwecke als besonders brauchbar erwiesen.
Wegen des großen Interesses bei den Lesern wurden die bereits in der ELEKTRONIK veröffentlichten Beiträge über PASCAL sorgfältig überarbeitet, ergänzt und mit weiteren bisher unveröffentlichten Arbeiten zu diesem Sonderheft zusammengefaßt. Damit soll allen, die sich in diese Programmiersprache einarbeiten möchten oder müssen, also Studenten, Ingenieure und Techniker, die mit Mikrocomputern arbeiten, Hilfestellung beim Einstieg sowie nützliche Informationen für die Praxis gegeben werden.

Die Redaktion

Inhalt

Vorwort 1

Inhalt 2

PASCAL 3

PASCAL im Vergleich mit anderen Programmiersprachen 8

Programmieren mit PASCAL 17

Ein Mikrocomputer – für PASCAL konzipiert 52

System-PASCAL – Programmiersprache mit abgerundetem Konzept 55

Welches PASCAL für welchen Computer? 60

Kurzberichte 63

Fachbücher 64

1980
Franzis-Verlag GmbH, Karlstraße 37, 8000 München 2. Produktion: ELVAG, Elektronik Verlag Luzern AG, CH-6002 Luzern.
Sonderheft der Zeitschrift ELEKTRONIK. Redaktion: Peter von Bechen, für den Text verantwortlich: Günther Klasche.

© Sämtliche Rechte – besonders das Übersetzungsrecht – an Text und Bildern vorbehalten. Fotomechanische Vervielfältigung nur mit Genehmigung des Verlages. Jeder Nachdruck, auch auszugsweise, und jede Wiedergabe der Abbildung, auch in verändertem Zustand, sind verboten.

ISSN 0170-0898
Druck: Stämpfli, CH-Bern
ZV-Artikel-Nr. 46021 · ST/ZV/880/329/6'

Dipl.-Ing. Franz J. Dorfner

PASCAL

Die höhere Programmiersprache PASCAL ist eine der wenigen in den letzten Jahren entwickelten Programmiersprachen, die eine weltweite Anerkennung erlangte. Bei der Konzipierung von PASCAL waren besonders zwei Entwicklungsziele von Bedeutung: Die Sprache soll klar und natürlich definiert sein, und beim Erlernen der Programmiersprache soll die Technik der strukturierten Programmierung unterstützt werden. Sie soll zugleich effizient und zuverlässig auf heutigen Rechenanlagen implementierbar sein. Gerade diese Entwurfsziele kommen der rasanten Entwicklung auf der Mikroprozessorebene entgegen. Hier sucht man seit Jahren nach übersichtlichen, klar strukturierten Softwareerstellungsmethoden. Es zeichnet sich auch eindeutig, besonders bei den 16-Bit-Prozessoren, der Trend zu höheren Programmiersprachen ab. Zur Einführung sollen hier die wesentlichen Merkmale der Programmiersprache PASCAL dargestellt werden.

1 Geschichte

Anfang der 70er Jahre kreierte *Niklaus Wirth*, Professor an der Technischen Hochschule in Zürich, die Programmiersprache PASCAL, die er nach dem französischen Philosoph und Mathematiker *Blaise Pascal* (1623...1662) benannte. *Urs Ammann* entwickelte dann, damals ebenfalls an der Technischen Hochschule in Zürich, zusammen mit *Nikolaus Wirth* und Mitarbeitern einen PASCAL-Compiler für die CDC 6000 von Control Data. Damit war der Startschuß für die Verbreitung dieser Programmiersprache gegeben. Mittlerweilen ist PASCAL schon auf über 50 Rechnern implementiert worden und setzt sich immer mehr als dominierende Programmiersprache durch. PASCAL läuft jetzt schon auf Mikroprozessorsystemen wie z. B. 8080, 9900 sowie auch auf Großrechnersystemen wie z. B. auf dem superschnellen Rechner Cray-1.

Es blieb natürlich nicht aus, daß bald Änderungen bzw. Erweiterungen an der Sprache vorgenommen wurden. So existieren nun verschiedene Versionen von PASCAL. Das Wirthsche PASCAL, welches in [1] beschrieben ist, bezeichnet man als Standard-Version. Betrachtet man die Aktivitäten der letzten Zeit, so kann man feststellen, daß sich besonders das sogenannte UCSD-PASCAL durchsetzt. (UCSD: *University of California at San Diego*). UCSD-PASCAL entstand unter Federführung von Professor *Kenneth Bowles* an der Universität in San Diego und hat einige Erweiterungen gegenüber dem Standard-PASCAL wie z. B. die Datentypen *String* und *Long Integer* sowie freien Zugriff auf Komponenten innerhalb eines Files mittels der Funktion *SEEK* (siehe Abschnitt 5).

2 Struktur von PASCAL

Für PASCAL hat ALGOL 60 Pate gestanden und ist wie diese eine blockstrukturierte Programmiersprache. Ein Programm besteht aus einem Programmkopf, einem Deklarationsteil und aus einem Anweisungsteil. Deklarationsteil und Anweisungsteil bilden zusammen einen Block. Der Programmkopf beginnt mit dem Symbol „PROGRAM", es folgen: der Programmbezeichner (Name des Programmes, der auf die Ausführung des Programmes keinen Einfluß hat) und die Programmparameter. Dies sind in der Regel Files, welche außerhalb des Programmes existieren und das Programm mit der Umwelt verbinden. Die Ein- und Ausgabe erfolgt über die Standardfiles: Input, Output. Ein Block, der den Deklarationsteil und den Anweisungsteil umfaßt, kann in sechs Abschnitte unterteilt werden (*Bild 1*). Die ersten vier Deklarationen sind: Markenvereinbarungsteil, Konstantenvereinbarungsteil, Typenvereinbarungsteil und Variablenvereinbarungsteil. Der fünfte Abschnitt ist der Prozedur-/Funktionsvereinbarungsteil. Der letzte Abschnitt ist der Anweisungsteil, der durch die Symbole „BEGIN" und „END" geklammert ist. Mehrere Anweisungen werden durch das Sonderzeichen „;" voneinander getrennt. Ein Programm wird ausgeführt, indem der Block ausgeführt wird. Das heißt, der Deklarationsteil wird realisiert und dann wird der Anweisungsteil abgearbeitet. In ein Programm können Kommentare in der Form (*Kommentar*) eingefügt werden. Diese Kommentare dienen der Lesbarkeit des Programmes und haben auf dessen Ablauf keinen Einfluß.

Eine Prozedur- bzw. Funktionsvereinbarung führt selbst wieder einen Block mit Deklarationsteil und Anweisungsteil ein.

PROGRAM Programmbezeichner
(Programmparameter); → Programmkopf
Markenvereinbarungsteil
Konstantvereinbarungsteil
Typvereinbarungsteil
Variablenvereinbarungsteil
Prozedur/Funktionsvereinbarungsteil
Anweisungsteil
} Block

Ein PASCAL-Programm hat die syntaktische Form:
PROGRAM Programmbezeichner (Programmparameter); Block.

Bild 1. Programmstruktur

```
PROGRAM A (...);
.....
    PROCEDURE B (...);
    .....
        PROCEDURE C (...);
        .....
        BEGIN .... END;
    BEGIN .... END;
    FUNCTION D (...);
    .....
    BEGIN .... END;
BEGIN .... END.
```

Level 0: A Level 1: B,D Level 2: C

Bild 2. Blockschachtelung

Es ergibt sich damit eine Blockschachtelungsmöglichkeit wie z. B. nach *Bild 2*. Die in einem Block vereinbarten Größen heißen *lokal* bezüglich dieses Blokkes. Alle übrigen Größen dieses Blockes heißen *global* und müssen in einem übergeordneten Block vereinbart sein. Jede Vereinbarung für eine Größe hat nur in demjenigen Block Gültigkeit, in dem sie vorgenommen wurde, und in jedem ihm untergeordneten Block.

3 Datentypen

Die Leistungsfähigkeit der Programmiersprache PASCAL resultiert vor allem auch aus der Vielfalt an Datentypen und daher, daß die Datenstrukturen nicht maschinen- sondern problemorientiert sind. Man unterscheidet drei Gruppen von Datentypen: die einfachen Typen, die strukturierten Typen und die Pointertypen.

3.1 Einfache Typen

Bei den einfachen Typen gibt es vier standardmäßig festgelegte Datentypen, dies sind: Integer (ganze Zahlen), Real (reelle Zahlen), Boolean (die logischen Werte True und False) und Char (ASCII-Zeichen). Sie bilden einen Grundbereich, dessen Wertebereich (eine Größe des Typs Boolean hat natürlich nur den Wertebereich True und False) maschinenabhängig ist. Zu den einfachen Typen zählen auch noch die Aufzählungstypen, bei denen durch eine Typendefinition im Deklarationsteil der Wertebereich festgelegt (aufgezählt) wird, z. B. Typ Richtung = (Süd, Nord, West, Ost). Andere Programmiersprachen wie ALGOL 60 oder FORTRAN kennen solche Aufzählungstypen nicht.

Es lassen sich auf Unterbereichstypen definieren, deren Wertebereich auf eine Teilfolge des Gesamtwertebereiches eingeschränkt ist, z. B. Typ Monat = 1...12. Damit bilden die Ziffern 1..12 den Wertebereich des Typs Monat. Diese Einschränkung trägt auch zur besseren Lesbarkeit des Programmes bei.

3.2 Strukturierte Typen

Strukturierte Typen werden definiert durch die Angabe des Typs ihrer Komponenten und deren Strukturierungsmethoden. Die verschiedenen Strukturierungsmethoden unterscheiden sich bezüglich des Zugriffes auf die Komponenten eines strukturierten Typs. Bei PASCAL hat man vier Strukturierungsmethoden zur Verfügung, dies sind: Array-Struktur, Record-Struktur, File-Struktur und Set-Struktur.

Bei einer Array-Struktur sind alle Komponenten vom selben Typ. Man kann damit ein- oder mehrdimensionale Felder abspeichern. Der Zugriff auf die Komponenten eines Feldes geschieht über Indizes. Mit „U = ARRAY [1..10] of Real" haben wir ein Feld U mit den zehn Komponenten U [1]...U [10] des Typs Real definiert. Um auf die Komponenten zugreifen zu können, führt man eine Variable vom Typ Integer ein, die als Index dient. Bei PASCAL hat man auch die Möglichkeit, als Index die Buchstaben des Alphabetes zu verwenden. Mit „M = Array [‚A'..‚Z'] of Integer" wird ein Feld mit 26 Komponenten des Typs Integer definiert. Die Variable für den Index ist hier vom Typ Char. Bei einem Array gibt es über die Indizes einen Random-Access (freien Zugriff) auf die Komponenten des Feldes.

Bei der Record-Struktur müssen die Komponenten nicht vom selben Typ sein. Ihnen werden Bezeichner zugeordnet, mit denen man auf sie zugreifen kann. Die Bezeichner der einzelnen Komponenten sind nicht berechenbar wie etwa die Indizes bei der Array-Struktur. PASCAL hat nicht den Datentyp Complex wie FORTRAN. Eine Abhilfe wird erreicht durch Einführung des Datentyps Complex als Record, z. B. „Complex = Record Re, Im: Real End". Mit einer Variablen Z vom Typ Complex können wir dann auf die Komponenten Z.Re und Z.Im zugreifen (siehe Programmbeispiel *Bild 3a*). Ein weiteres Beispiel eines Records: Zeit = Record Stunde: 1...24; Minute, Stunde: 1...60 End.

Eine Set-Struktur ist eine Mengendefinition, deren Wertebereich die Potenzmenge des Basistyps ist. Bei A = SET OF B mit dem Wertebereich von B, $(B_0, ..., B_{n-1})$. Dann ist der Wertebereich des Mengentyps A die Potenzmenge 2^B des Typs B, er enthält die 2^n Mengen:

( )	leere Menge,
(B_0)	1-elementige Menge,
(B_0, B_1)	2-elementige Menge,
.....	
$(B_0, ..., B_n)$	n-elementige Menge,

wobei ein Wert B_i höchstens einmal in einer Menge vorkommt. Mit der Variablenvereinbarung VAR X:Set of 1..3; kann die Variable X vom Typ Menge einen der Werte [ ], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3] an-

nehmen. Mit Variablen vom Set-Typ können die Mengenoperationen wie etwa Vereinigung und Durchschnitt realisiert werden.

Die File-Struktur ist eine Sequenz von Komponenten desselben Typs. Der Zugriff auf die Komponenten des Files geschieht sequentiell lesend bzw. sequentiell schreibend. Ein direkter Zugriff (Random-Access) ist hier nicht möglich. Bei der Filedeklaration ist die Anzahl der Komponenten und damit die Länge des Files nicht festgelegt. Dies ist auch ein wesentlicher Unterschied zu einem Array. Die Vereinbarung einer Variablen mit File-Struktur, z. B. X = File of real schließt automatisch die Vereinbarung einer Buffer-Variablen X↑ des Typs Real ein. Die Buffer-Variable ist ein Fenster in der sequentiellen Anordnung des Files und enthält die jeweils aktuelle Komponente des Files. Für die Fileende-Kennung, wie für den Schreib- und den Lesevorgang existieren die folgenden Standardprozeduren:

```
PROGRAM COMMUL (OUTPUT);

(*Multiplikation der komplexen Zahlen (1+j2)*(2–j1)*)
TYPE COMPLEX = RECORD RE,IM : INTEGER END;
VAR X,Y,Z : COMPLEX;
BEGIN
    X.RE:=1; X.IM:=2;
    Y.RE:=2; Y.IM:=–1;
    Z.RE:=X.RE*Y.RE–X.IM*Y.IM;
    Z.IM:=X.RE*Y.IM+X.IM*Y.RE;
    WRITELN('Produkt (1+j2)*(2–j1)');
    WRITELN('        Realteil = ',Z.RE);
    WRITELN('Imaginaerteil = ',Z.IM);
END.
```

Bild 3a. Programmbeispiel Multiplikation komplexer Zahlen

```
PROGRAMM ACKER (INPUT, OUTPUT);

(*Die Ackermann-Funktion wird häufig für den Geschwindigkeits-
vergleich von Rechenanlagen verwendet. Die Funktion ist stark re-
kursiv und kann sehr lange Ausführungszeiten zur Folge haben. Die
Definitionsgleichungen lauten:

I)  F(O,Y) =Y+1;
II) F(X,O) =F(X–1,1);
III) F(X,Y) = F(X–1,F(X, Y–1));
Die Ausführungszeit für die Berechnung von F(3, 4) beträgt auf der
PASCAL-MICROENGINE nur zwei Sekunden. Der Funktionswert
von F(3,4) ist 125.*)

VAR DUMMY: CHAR;
X, Y:          INTEGER;
FUNCTION F (X,Y:INTEGER) : INTEGER;
BEGIN
    IF X=O THEN F:=Y+1
            ELSE IF Y=O THEN F:=F(X–1,1)
                        ELSE  F:=F(X–1,F(X,Y–1));
END.
BEGIN
    REPEAT
        WRITELN;WRITELN;
        WRITELN('Ackermann-Funktion F(X, Y)');
        WRITE('X='); READLN(X);
        WRITE('Y='); READLN(Y);
        WRITELN;
        WRITELN('F(',X,',',Y,')=',F(X,Y));
        WRITELN;
        WRITE('ENDE ?      (J/N)');
        READ(DUMMY);
    UNTIL DUMMY='J';
END.
```

Bild 3b. Programmbeispiel Ackermann-Funktion

eof(x): Boolesche Funktion für die Fileende-Erkennung. Wenn die Buffer-Variable X↑ hinter dem *end of the file* X steht, nimmt eof(x) den Wert True an.

reset(x): Hiermit wird der File X für sequentielles Lesen eröffnet. Der Buffer-Variablen X↑ wird die erste Komponente zugewiesen und eof(x) bekommt den Wert „False".

get(x): Mit dem Prozeduraufruf get(x) rückt der interne Zeiger vor auf die nächste Komponente des Files X und weist der Buffer-Variablen X↑ den neuen Wert zu. Damit kann der File X sequentiell gelesen werden.

rewrite(x): Hiermit wird der File X für sequentielles Schreiben eröffnet. Die Buffer-Variable X↑ wird auf die erste Komponente des Files gesetzt und eof(x) bekommt den Wert „True".

put(x): Nach einer Zuweisung im Programm an die Buffer-Variable X↑, bewirkt der Prozeduraufruf p(x), daß die aktuelle Komponente des Files X den Wert von X↑ erhält. Damit kann auf den File X sequentiell geschrieben werden.

3.3 Pointer-Typen

Eine im Deklarationsteil eines Blockes vereinbarte Variable ist eine statische Variable, da nach der Vereinbarung sowohl der symbolische Name als auch der zugeordnete Speicherplatz während der Ausführung eines Blockes festliegt. Mit der Vereinbarung einer Pointervariablen P kann eine dynamische Variable des angegebenen referierenden Variablentyps eingeführt werden. Diese generierte Variable erhält den symbolischen Namen P↑ und wird mit dem Aufruf der Standardprozedur „NEW" erzeugt. Bei jeder Vereinbarung einer Pointervariablen P des referierenden Variablentyps T wird automatisch eine Variable P↑ des Typs T vereinbart. Ihr Wert ist der dynamische Wert, auf den P zeigt. Jede Pointervariable kann unabhängig von ihrem referierenden Typen den Wert „NIL" annehmen. Dies besagt, daß die Pointervariable auf keine Variable verweist. Sei z. B. P eine Pointervariable des referierenden Variablentyps Integer, so ist folgende Zuweisung an die generierte dynamische Variable möglich:

```
VAR P: ↑Integer;
...
NEW(P); P↑: = 128;
```

Pointerstrukturen sind ein effizientes Hilfsmittel, um komplexe und flexible Datenstrukturen aufzubauen.

4 Vorteile der Programmiersprache PASCAL

Gerade von Leuten, die in der Vergangenheit mit anderen Programmiersprachen zur vollen Zufriedenheit programmiert haben, kommt die Frage, welche Vorteile die neue Sprache dem Anwender bietet. Im Folgenden sind einige Argumente für PASCAL aufgeführt:

- Unterstützt das strukturierte Programmieren,
  weil ein Programm in viele abgeschlossene Einheiten zerlegt werden kann; die Programme statisch überprüft werden können; ein Programm gut lesbar ist und damit schnell verstanden werden kann; weil Kontrollstrukturen existieren, usw.
- Einfache und übersichtliche Sprache,
  weil nicht alle sinnvollen, sondern alle notwendigen Sprachkonstruktionen verwendet wurden. Damit ist die Komplexität und zum Teil Unübersichtlichkeit von PL/1 und ALGOL 68 vermieden worden.
- Klare, exakte und vollständige Sprachdefinition
  Zur syntaktischen Beschreibung von PASCAL verwendet man Syntaxdiagramme wie in *Bild 4* für einen Block.
- Trennung von Vereinbarungs- und Anweisungsteil
  Alle im Anweisungsteil verwendeten Objekte müssen im Vereinbarungsteil vereinbart sein; der Anweisungsteil ist mit BEGIN und END geklammert.
- Problemgerechte Beschreibung von Datentypen
  Problemadäquate Datenbeschreibung, im Gegensatz zu ALGOL 60 z. B.

  PASCAL
  Feld = ARRAY [1..5] OF INTEGER;

  ALGOL 60
  INTEGER ARRAY Feld [1..5]
- Problemangepaßte Beschreibung von Datentypen
  Möglichkeit, durch die Typenvereinbarung eigene Datentypen zu definieren. Unterbereichstypen erhöhen die Lesbarkeit eines Programmes.
- Kontrollstrukturen
  Blockzusammenfassung BEGIN...END; Auswahl IF...THEN...ELSE, CASE...OF...END; Wiederholung WHILE...DO, REPEAT...UNTIL, FOR...TO...DO;
- Rekursiver Aufruf von Prozeduren und Funktionen
  Die Möglichkeit, daß innerhalb einer Prozedur oder Funktion diese selbst wieder aufgerufen werden kann (siehe Programmbeispiel Ackermann-Funktion *Bild 3b*).

5 Unterschiede zwischen Standard-PASCAL und UCSD-PASCAL

Wie eingangs schon erwähnt, haben sich bei der Programmiersprache PASCAL bereits verschiedene Dialekte gebildet. Dies wurde auch gerade dadurch gefördert, weil einige, für eine moderne Programmiersprache sogar unabdingbare Merkmale im Standard-PASCAL fehlen. Das UCSD-PASCAL, welches auf der „PASCAL P-Code Maschine" der Spezial Electronic, München, implementiert ist, ist mit einer Reihe von Erweiterungen bzw. Unterschieden gegenüber dem Standard-PASCAL ausgestattet. Diese Erweiterungen geben dem Softwareentwickler mehr Möglichkeiten, und er erreicht damit eine schnellere und effektivere Lösung seines Softwarevorhabens. Die nachfolgende Auflistung soll diejenigen, die sich bereits mit Einzelheiten der Programmiersprache PASCAL beschäftigt haben, mit den Unterschieden vertraut machen.

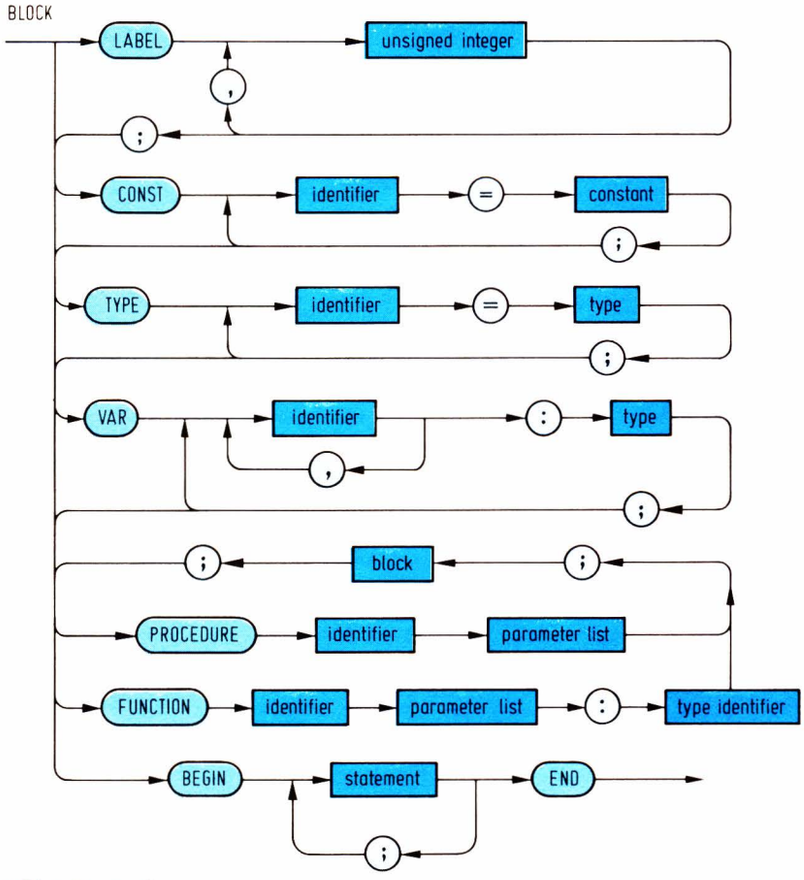


Bild 4. Syntaxdiagramm

5.1 *Case-Statement*

Die Case-Anweisung ist eine Fallunterscheidung, bei der mittels eines Selektors verschiedene Anweisungen aufgelöst werden können. Kommt der Wert des Selektors nicht in der Fallunterscheidungsliste vor, so führt dies beim Standard-PASCAL zu einem unliebsamen Abbruch des Programmes. Beim UCSD-PASCAL führt diese Situation nicht zum Abbruch, es wird das nächste Statement nach dem zum Case-Statement gehörendem End ausgeführt.

5.2 *Dynamic Memory Allocation*

In Standard-PASCAL gibt es die Prozedur DISPOSE (P). Damit wird der Speicherplatz im Heap, der mit NEW (P) für die dynamische Variable P↑ angelegt wurde, wieder frei. UCSD-PASCAL hat nicht die Prozedur DISPOSE, an deren Stelle gibt es die Prozeduren MARK UND RELEAS, die eine optimale Heapverwaltung ermöglichen.

5.3 Comments

Der UCSD-Compiler registriert ein $-Zeichen direkt nach dem Kommentarzeichen als eine spezielle Anweisung an den Compiler. Beispielsweise ist (* $ L + Console: *) eine Anweisung an den Compiler, ein Compilerlisting zu erstellen und dies auf der Console auszugeben. (* $ G-*) heißt für den Compiler, daß die Sprunganweisung GOTO im Programm nicht erlaubt ist. In Standard-PASCAL sind solche Compiler-Optionen nicht möglich.

5.4 Segment Procedures

UCSD-PASCAL ermöglicht die Segmentierung von Programmen. Prozeduren und Funktionen werden hierbei nur dann in den Arbeitsspeicher geholt, wenn sie vom Programm aufgerufen werden. Standard-PASCAL hat diese Möglichkeit nicht.

5.5 Interactive Files

UCSD-PASCAL erweitert das Standard-PASCAL im Bereich des Filehandlings um den Filetyp INTERACTIVE. Dieser Typ gibt dem Programmierer mehr Flexibilität bei der Input-/Outputprogrammierung mit interaktiven Peripheriegeräten. Im UCSD-PASCAL sind die beiden Standard-Files INPUT und OUTPUT vom Typ INTERACTIVE.

5.6 Random Access Disk Files

Standard-PASCAL hat nicht die Möglichkeit des wahlfreien Zugriffes auf Records in einem Disk-File. UCSD-PASCAL hat hierfür die Prozedur SEEK, mit der der Wert des Filepointers auf eine bestimmte Recordnummer gesetzt werden kann.

5.7 GOTO and EXIT

Im Standard-PASCAL ist auch der Sprung auf eine Marke außerhalb des Blockes, in welchem das GOTO-Statement steht, möglich. Dies kann zu einer unübersichtlichen Programmierung führen und wurde daher im UCSD-PASCAL nicht übernommen. Es sind nur Sprünge innerhalb eines Blockes möglich. Ein Block kann mit der Prozedur EXIT verlassen werden. Dies ist identisch mit einem GOTO zu einer Marke nach dem letzten Statement des betreffenden Blockes.

5.8 Program Header

Im Standard-PASCAL beinhaltet der Programmkopf eine Liste von Files in der Form von Parametern. UCSD-PASCAL akzeptiert zwar diesen Programmkopf, die Angabe der Filenamen ist jedoch nicht erforderlich.

5.9 String Variables

UCSD-PASCAL hat den Typ STRING für das Arbeiten mit einer Sequenz von Charaktern vorgesehen. Eine Stringvariable ist identisch mit: Packed Array [1..obere Grenze] of CHAR inklusive einem Längenbyte. Stringvariable im UCSD-PASCAL werden unterstützt durch die Prozeduren INSERT und DELETE und durch die Funktionen LENGTH, POS, COPY und CONCAT. Jeder Programmierer wird diese Erweiterungen des UCSD-PASCAL besonders zu schätzen wissen. Im Standard-PASCAL existiert der Typ STRING nicht.

5.10 Sets

UCSD-PASCAL limitiert die Anzahl der Elemente einer Variablen des Typs SET auf höchstens 4080. Andere PASCAL-Implementationen limitieren die Zahl der Elemente auf 64 oder noch weniger. Standard-PASCAL macht keine Aussage, wieviele Elemente ein SET haben darf.

5.11 Long Integers

UCSD-PASCAL ermöglicht das Arbeiten mit Integergrößen von bis zu 36 Dezimalstellen. Der Programmierer kann die Anzahl der benötigten Dezimalstellen einer Integervariablen im Deklarationsteil angeben.

5.12 Separately Compiled Units

UCSD-PASCAL ermöglicht die separate Compilierung einer Einheit von Prozeduren, Funktionen und den dazugehörigen Deklarationen in eine UNIT. Diese UNIT kann dann von anderen Programmen benutzt werden, ohne sie wieder compilieren zu müssen. Standard-PASCAL hat nichts entsprechendes zu bieten.

Diese Erweiterungen bzw. Änderungen machen PASCAL noch leistungsfähiger und bilden auch die Voraussetzung für einen breiten industriellen Einsatz dieser neuen Programmiersprache. Es ist natürlich nicht wünschenswert, wenn sich zu viele Dialekte bilden. Momentan bemüht man sich noch auf internationaler Ebene um eine Standardisierung von PASCAL. Bis jetzt kann aber noch nicht gesagt werden, welche bzw. wieviele Erweiterungen von diesem Gremium in den neuen Standard aufgenommen und abgesegnet werden.

Literatur

1 *Jensen, K., Wirth, N.:* PASCAL, User Manual and Report. Springer-Verlag, New York, Heidelberg, Wien.

2 *Rohlfing, H.:* PASCAL, Eine Einführung. BI Hochschultaschenbücher, Band 756.

3 *Schauer, H.:* PASCAL für Anfänger. Oldenbourg Verlag, Wien, München.

4 *Schauer, H.:* PASCAL Übungen. Oldenbourg Verlag, Wien, München.

5 *Kaucher, Klatte, Ullrich:* Höhere Programmiersprachen ALGOL, FORTRAN, PASCAL. BI Wissenschaftsverlag, Reihe Informatik, Band 24.

6 Skriptum zur PASCAL-Vorlesung für die DV-Grundausbildung an der Technischen Universität Berlin.

7 PASCAL MICROENGINE Reference Manual, The Microengine Company.

8 *Kenneth L. Bowles:* Beginner's Guide for the UCSD PASCAL System. McGraw Hill, USA.

Dipl.-Ing. Franz J. Dorfner wurde in Passau geboren. Er studierte an der Fachhochschule in Regensburg und wechselte danach an die Technische Universität Berlin. Dort schloß er 1978 das Studium für Nachrichtentechnik ab. Anschließend ging er zur Spezial Electronic nach München, wo er jetzt für die Mikroprozessor- und Mikrocomputer-Systemlinie zuständig ist.

Hobbys: Skifahren, Lesen, Wandern

Telefon: (0 89) 53 03 87

ELEKTRONIK-Leser seit 1970

Kenneth Wickham

PASCAL im Vergleich mit anderen Programmiersprachen

Seit einiger Zeit ist die höhere Programmiersprache PASCAL bei vielen Mikro- und Minicomputeranwendern im Gespräch. Man verspricht sich von ihrem Einsatz eine Kostensenkung bei der bisher überproportionalen teuren Software.

Ein großes amerikanisches Halbleiter-Unternehmen hat sich schon frühzeitig mit PASCAL befaßt und sich entschieden, sie als Standard-Programmiersprache für den internen Gebrauch zu benutzen. Aufgrund dieser Erfahrungen versucht der Autor diese Sprache mit anderen zu vergleichen.

Die stark zunehmende Tendenz, Hardware-Logik durch programmierbare Mikroprozessoren zu ersetzen, zwingt die Elektronikentwickler, nach Wegen zu suchen, wie man kostengünstiger Programme schreiben, warten und überarbeiten kann. Programmierer waren schon immer der Meinung, daß die Verwendung einer höheren Programmiersprache der effektivste Weg sei, die Softwarekosten zu senken, was besonders auf eine Sprache wie PASCAL zutrifft, die eine sog. „*Top-down*"-Struktur bei der Programmerstellung ermöglicht.

Wenn ein Programmierer vergleicht zwischen dem Schreiben in PASCAL oder in der Assemblersprache eines Mikroprozessors, so ergibt sich, daß die Programmiereffizienz um den Faktor 2,5...3 besser ist und die Zeit für die Programmentwicklung und -wartung um den Faktor 2 verringert wird.

Weil man keine allgemeine Aussage darüber machen kann, was an PASCAL besser ist als bei anderen höheren Programmiersprachen, ist in *Bild 1* der Versuch unternommen worden, fünf Sprachen in ihren wichtigsten Kriterien zu vergleichen. Daraus und der ausführlichen Übersicht in *Tabelle 1* ergibt sich, daß PASCAL ganz klar die Sprache ist, die man wählen sollte, wenn es auf Kostenminimierung für die Software während ihrer gesamten Nutzungsdauer ankommt.

Der Schritt vom Gebrauch der Maschinensprache zu PASCAL erhöht die Produktivität aus drei Gründen:

1. Jeder Befehl in der höheren Sprache erzeugt mehrere Maschinenbefehle. Dadurch kann der Programmierer das Problem auf einer abstrakten Ebene lösen und muß sich nicht mit der Hardwarestruktur der Maschine beschäftigen.
2. Eine strukturierte Sprache wie PASCAL erlaubt, Probleme so zu lösen, wie der natürliche Denkprozeß verläuft – von der Gesamtstruktur zur Teilfunktion („*Top down*"). Daraus ergeben sich weniger Fehler und falls solche auftreten, sind sie einfacher zu finden.
3. Weil PASCAL leichter lesbar ist, sind Änderungen ohne große Schwierigkeiten leicht durchzuführen.

Anders als Hardware, die sich nur mühsam verändern läßt, wird Software kontinuierlich modifiziert, um sie den Anforderungen des Benutzers anzupassen. Deshalb ist es wichtig, nicht nur die Entwicklungskosten, sondern auch die Änderungskosten möglichst gering zu halten. Bei Anwendungen, die in hohem Maße den Dialog Mensch–Maschine erfordern, oder bei denen die Problemstellung sich laufend ändert, wie z. B. bei der Sammlung von Management-Daten, können die Kosten für die Wartung jährlich über 20 % der Entwicklungskosten liegen.

Die Benutzung von PASCAL hat ähnlich wie andere höhere Programmiersprachen nicht nur Vorteile:

- Ein PASCAL-Programm erfordert, wenn es compiliert wird, etwa doppelt soviel Speicherplatz wie ein Programm in der Maschinensprache.
- Das in PASCAL geschriebene Programm wird nicht so schnell ausgeführt wie ein Maschinenprogramm.

In Zukunft wird zusätzlicher Speicherplatz jedoch nicht mehr so teuer wie heute sein. Nach der bisherigen Erfahrung sind die Preise für Speicherbausteine alle 3 Jahre um 50 % gesunken.

Heute können PASCAL und andere höhere Programmiersprachen auf Mikroprozessorsystemen laufen (nicht übersetzt werden), die nicht mehr als 8 KByte Speicherplatz besitzen.

Mikroprozessoren mit 8 KByte werden heute als mittlere Systeme betrachtet. Da es die technische Entwicklung in Zukunft erlauben wird, größere Speicherkapazitäten auf ein Chip zu integrieren, wird sicherlich in Zukunft PASCAL auch für kleinere Systeme nichts Ungewöhnliches sein.

Neuartige Mikroprozessoren, Mikrocomputer, Peripheriebausteine und Speicherchips ersetzen immer wieder bisher bekannte Typen. Die modernen Bauteile mit höherer Kosteneffektivität erfordern wegen architektonischer Veränderung, besonders vom Befehlssatz her, eine neue Software. Jeder neue Prozessortyp muß mit Software zur Entwicklung unterstützt werden (Assembler, Utilities, Betriebssystem), wodurch der Programmierer immer wieder gezwungen ist, neue Assemblersprachen-Mnemonics und Programmiertechniken zu lernen. Die Schwierigkeiten, die sich für den Softwarebereich daraus ergeben, könnten erheblich verringert werden, wenn man eine maschinenunabhängige, transportable Sprache verwenden würde – d. h. eine Sprache und das dazugehörige Unterstützungspaket, das für verschiedene Architekturen geeignet ist, auch für solche, die noch entwickelt werden.

Die Originalform von PASCAL, die *Niklaus Wirth* entwickelt hat, besitzt bereits die meisten Sprachei-

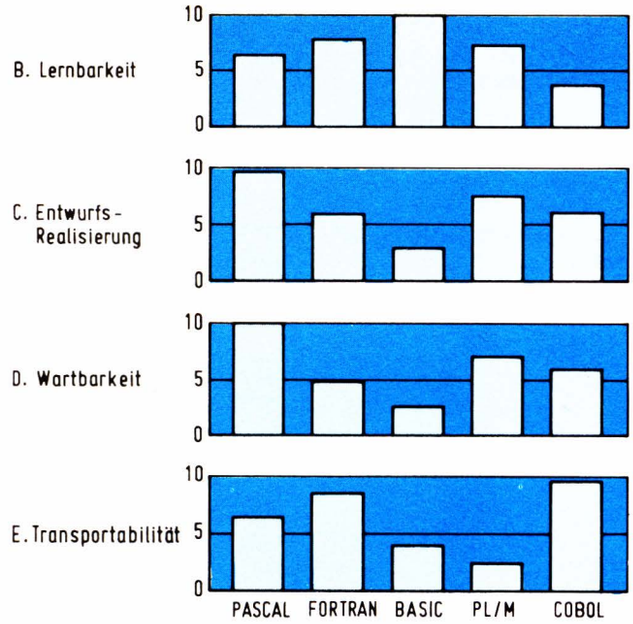


Bild 1. Die fünf Sprachen aus Tabelle 1 gewichtet nach der Wertigkeit der vier Phasen der Software-Lebensdauer. Die erste Phase (A) benötigt keine Software

genschaften, die für leichte Transportabilität sowie Wartbarkeit notwendig sind.

Hauptmerkmal ist die Flexibilität, Daten zu beschreiben. Der Programmierer kann freizügig Datentypen definieren und der Compiler legt dann seine eigene variable Datenzuweisung fest. Dadurch wird verhindert, daß der Programmierer Tricks anwendet, um Datentypen festzulegen, wie das bei FORTRAN oft getan wird, um die Möglichkeit zur Textverarbeitung zu erhalten. Solche Tricks machen die Transportabilität unmöglich.

Ein weiterer Grund für die Transportabilität, der auch die Erstellung von Reentrant-Code unterstützt, ist, daß es im Standard-PASCAL keinen einfachen Weg gibt, den Run-Time-Codebereich zu adressieren. Anders gesagt: es ist schwierig, einen Code zu schreiben, der sich selbst ändert. Reentrant-Code gestattet mehrfache Abarbeitung, die gleichzeitig von einer Kopie des Codes aus erfolgt. Diese Eigenschaft ist wichtig für den Entwurf von effizienten Echtzeit-Systemen mit parallel ablaufenden Operationen. Der „Mikroprozessor PASCAL"-Compiler von Texas Instruments erzeugt Reentrant-Code. Die Mikroprozessor-PASCAL-Executive-Library unterstützt darüber hinaus die Entwicklung von Reentrant-, Concurrent-Programmen, deshalb müssen die Routinen nicht kopiert werden, wenn sie für verschiedene Tasks gleichzeitig gebraucht werden.

Die gut definierte und klare E/A-Struktur erleichtert ebenfalls das Problem, ein transportables Software-Interface zu entwerfen, mit dem auch nach hardwareseitigen Änderungen weiter gearbeitet werden kann. Beim Mikroprozessor-PASCAL wird allerdings etwas von der klaren Struktur geopfert, um eine bessere E/A-Flexibilität zu erreichen. Texas Instruments machte einige Modifikationen an PASCAL, aber die allgemeinen Grundsätze sind bei der Originalversion von *N. Wirth* und der Version von TI gleich.

Mischung von PASCAL und Assembler-Code

Die Assemblersprache ist eigentlich der beste Weg, um Programme, die schnell ausgeführt werden müssen, optimal auszuarbeiten, obwohl man keine Wartbarkeit und Transportabilität wie in einem Programm in höherer Sprache hat. Das Mikroprozessor-PASCAL-System enthält einen Reverse-Assembler, mit dem man Routinen, die in PASCAL geschrieben sind, in die Assemblersprache umsetzen und für zeitkritische Anwendung optimieren kann.

Warum schreibt man dann diese Programmabschnitte nicht direkt in der Assemblersprache?

Man hat herausgefunden, daß die anfängliche Codierung und Fehlerbeseitigung in PASCAL mit nachträglicher „Optimierung von Hand" in Assemblersprache, wo notwendig, signifikante Vorteile hat:

- Beibehaltung der Möglichkeit des „Top-Down-Design",
- schnellerer Entwurf von Prototypen,
- zuverlässiger endgültiger Code.

Der Prototypen-Entwurf ist einfacher, weil Algorithmen leichter in PASCAL als in Assembler-Code zu schreiben sind.

PASCAL und die anderen höheren Programmiersprachen

PASCAL mit anderen höheren Programmiersprachen vergleichen, wie es in Tabelle 1 dargestellt ist, kann man eigentlich nur, wenn man Programmierexperten befragt, die alle Sprachen beherrschen.

Zu wissen, welche Sprache für eine bestimmte Aufgabe am besten geeignet ist, ist dann notwendig, wenn man von der Maschinensprache auf eine höhere Programmiersprache oder von einer solchen auf eine andere übergeht. Dieser Übergang kann zu großen Problemen führen, wenn ein grundsätzliches Redesign eines Softwareprodukts in Erwägung gezogen wird, oder wenn der Programmierer eine neue Sprache lernen muß. Deshalb sollten die Fakten, die in Tabelle 1 dargestellt sind, sorgfältig abgewogen werden:

- Wie an Punkt 3 zu ersehen ist, ist PASCAL nicht so einfach erlernbar wie z. B. BASIC, das allgemein als die Sprache gilt, die man besonders leicht lernen kann.

- Der Schwierigkeitsgrad beim Erlernen einer zweiten Programmiersprache (Punkt 4 und 5) hängt in gewissem Grad von der Sprache ab, die man zuerst gelernt hat. Sequentielle Sprachen wie BASIC oder FORTRAN erfordern einen anderen Denkprozeß als eine blockstrukturierte Sprache wie PASCAL. Dieser fundamentale Unterschied macht den Übergang von einem Sprachtyp zu einem anderen relativ schwer.

Bevor man die anderen Faktoren der Tabelle untersucht, muß man wissen, wie wichtig sie jeweils für jede Phase der gesamten Software-Lebensdauer sind.

Diese Phasen sind:

A. Problemanalyse,
B. Lernen die Programmiersprache anzuwenden,
C. Realisierung des Entwurfs,
D. Wartung der Software,
E. Wiederverwendung der Software nach Veränderungen oder Ersatz der Hardware.

Welche Wertigkeit die einzelnen Phasen haben, geht aus *Tabelle 2* hervor. Wenn man die Faktoren in Tabelle 1 damit gewichtet, ergibt sich das Balkendiagramm in Bild 1, die dann ein Maß für die „Güte" einer höheren Programmiersprache über ihre gesamte Lebensdauer darstellt.

Noch ein Wort zum Ursprung der Werte in der Tabelle, die ja eigentlich subjektive Abschätzungen sind: Der Autor sammelte die Daten in Tabelle 1, indem er etwa zwei Dutzend Programmierexperten befragte. Dabei wurden die Faktoren der Tabelle 1 drei verschiedenen Programmiergruppen vorgelegt: Spezialisten für kommerzielle Datenverarbeitung, Spezialisten für wissenschaftliche Programmierung und solchen für Echtzeitverarbeitung. Die Feststellung des Ergebnisses erfolgte so, daß für jede Sprache eine Bewertung aller 16 Faktoren abgegeben werden mußte. Nach teilweise langen Diskussionen wurden die Ergebnisse der drei Gruppen zur Tabelle 1 verarbeitet. Danach machte der Autor aufgrund seiner eigenen Erfahrung eine Tabelle mit Gewichtungsfaktoren (Tabelle 2). Softwareexperten, die andere Lebensabschnitte wählen, können dort auch ihre eigenen, für sie gültigen Faktoren einsetzen.

In der Analysephase (A) werden im wesentlichen die Hardware-Interface-Spezifikationen festgelegt sowie das Flußdiagramm festgelegt. Außerdem muß eine Softwarestruktur entwickelt werden, die die erforderliche Hardware unterstützt. Wenn völlig neuartige Softwareprodukte entwickelt werden sollen, sollte das zu einer natürlichen Auswahl der am besten geeigneten Programmiersprache für die Softwareentwicklung führen, ähnlich der Auswahl der Hardwarekomponenten. Die Analysephase erfordert selbst noch keine Softwaresprache.

Die zweite Phase, das Lernen (B), wurde bereits besprochen. In der dritten Phase, der Realisierung des Entwurfs (C), wird eine Sprache erforderlich, die das Top-Down-Design unterstützt. Diese Sprache sollte eine modulare oder strukturierte Annäherung an das Gesamtsystem unterstützen, d. h. beim Allgemeinen beginnend (*top*) und dann weiter zum Speziellen übergehend (*down*).

PASCAL stellt wegen der Blockstruktur das geeignete Mittel zur strukturierten Programmierung dar. Im Gegensatz dazu kann FORTRAN nicht ohne „GO-TO"-Programmsprünge geschrieben werden, die die Top-Down-Entwicklung verhindern. PASCAL hat auch „GO-TO"-Befehle, aber die Sprache ist so aufgebaut, daß man diese nicht benutzen muß.

Die vierte Phase im Lebenszyklus der Software, die Wartung (D), beginnt mit der Vervollständigung des ursprünglichen Entwurfs und erstreckt sich über mehrere Jahre. Softwarefehler nehmen normalerweise exponentiell mit der Zeit ab.

Die Korrektur von Fehlern oder das Einfügen von Softwareerweiterungen wird oft von Programmierern vorgenommen, die nicht die ursprünglichen Entwickler sind. Deshalb müssen Programme leicht verständ-

Tabelle 1. Subjektiver Vergleich von PASCAL mit vier anderen höheren Programmiersprachen

	PASCAL	FORTRAN	BASIC	PL/M	COBOL
1. Anwendbarkeit auf große Programme	10	6	3	8	7
2. Anwendbarkeit auf kleinere Programme	8	7	10	8	3
3. Lernbarkeit als erste Sprache	6	8	10	7	4
4. Lernbarkeit als zweite Sprache, wenn die erste blockstrukturiert war	8	6	10	9	3
5. Lernbarkeit als zweite Sprache, wenn die erste nicht blockstrukturiert war	5	9	10	6	4
6. Top-Down-Design	10	5	3	7	4
7. Unterstützung der strukturierten Programmierung	10	4	4	10	6
8. Schreibbarkeit von langen Programmen	10	5	3	9	5
9. Erzeugung von effizientem Code	8	7	6	10	7
10. Flexibilität von Datentypen	10	5	1	6	7
11. Überprüfung durch den Compiler auf logische Fehler	10	5	4	6	7
12. Gesteuerter Datenzugriff	10	6	1	3	4
13. Unterstützung effizienter Problemlösung	10	6	1	8	7
14. Lesbarkeit bei großen Programmen	10	4	2	8	8
15. Transportabilität	10	7	4	2	9
16. Effektivität von Standards	3	10	4	3	10

Tabelle 2. Die Punkte, die in Tabelle 1 aufgeführt sind, lassen sich bestimmten Lebensabschnitten der Software zuordnen

A	B	C	D	E
0	3, 4, 5	6...13	6...14 außer 9	15, 16

lich sein. Tatsächlich wirken sich viele Eigenschaften einer guten Programmiersprache, die vorteilhaft für die Realisierungsphase waren, auch positiv auf die Wartung aus. Lesbarkeit steht dabei an der Spitze der Liste, aber auch Top-Down-Design (6), Unterstützung der strukturierten Programmierung (7), Schreibbarkeit (8), Flexibilität der Datentypen (10), Compiler-Verträglichkeitsüberprüfungen (11) und die Steuerung des Datenzugriffs (12) sind ebenfalls wichtig. Der Einfachheit halber wurde jedem Faktor die gleiche Gewichtung gegeben (Tabelle 2).

Während der Nutzungsdauer eines Produkts kommt der Zeitpunkt, an dem alte Hardware durch neue, leistungsfähigere ersetzt werden muß. Diese fünfte Phase (E) kann ein völliges Überarbeiten von Hard- und Software bedeuten. Ideal wäre es natürlich, wenn man die Software ohne oder nur mit geringen Änderungen für die neue Hardware verwenden könnte. Ein maschinenunabhängiger Quellencode wäre die Lösung. Zur Zeit gibt es jedoch keine echte maschinenunabhängige Programmiersprache, obwohl PASCAL unter diesem Aspekt entwickelt wurde. Das Konzept, nach dem *N. Wirth* bei der Entwicklung von PASCAL vorging, war, die Sprache für eine theoretische Computerarchitektur, die von den meisten Computern als Subset realisiert werden kann, zu entwickeln.

Aber das Maß, wie transportabel eine Programmiersprache ist, wird auch durch den Standardisierungsgrad in der Industrie bestimmt. Während die Standardisierung weitgehend von der Reife einer Sprache und ihrer Aufnahme durch den Anwender bestimmt wird, kann sie auch durch die ursprüngliche Definition behindert werden. Wie man aus Tabelle 1 entnehmen kann, besitzt PASCAL eine hohe theoretische Transportabilität (15) aber hat zur Zeit einen geringen Standardisierungsgrad (16). Dies wird sich hoffentlich in naher Zukunft ändern, da ernsthafte Bestrebungen zur Formulierung eines Standards unternommen werden.

Vergleich der einzelnen Punkte

Viele der einzelnen Punkte in Tabelle 1 betreffen unterschiedliche Phasen in der Nutzungsdauer, und so scheint es sinnvoll, diese Punkte einzeln zu analysieren.

Große Programme leicht zu lesen und zu schreiben, (Punkt 1, 8 und 14) sind wichtige Eigenschaften in der Entwicklungsphase. Während die meisten Sprachen für kleine Programme (Punkt 2) geeignet sind, bieten doch nur wenige Lesbarkeit und Schreibbarkeit bei größeren Programmen. Die grundlegende Idee der Lesbarkeit ist, daß die Befehle beim Lesen leicht zu verstehen sind und daß die Aufgabenstellung des Programms leicht erkennbar ist. Die Schreibbarkeit hängt von der Klarheit und Kürze ab, mit der ein Programm aufgebaut wird.

Die Effizienz des vom Compiler erzeugten Codes (Punkt 9) ist der Punkt, der von Programmierern am meisten beim Vergleich von Sprachen diskutiert wird. Obwohl es stimmt, daß höhere Programmiersprachen im Gegensatz zur Assemblersprache „uneffizienten" Code produzieren, gilt das heute nicht mehr als wichtiger Faktor für Anwendungen mit geringen Stückzahlen, bei denen die Softwarekosten vergleichsweise hoch sind. Wie aus Tabelle 1 zu erkennen ist, ist PL/M bezüglich der Speichereffizienz etwas besser als PASCAL. Ein Grund dafür liegt in der Maschinenabhängigkeit von PL/M.

Es gibt z. B. PL/M-Codesequenzen, die es einem erlauben, zu einer absoluten Speicheradresse zu springen. Diese Möglichkeit ist bei Mikroprozessor-PASCAL verboten.

Gerade diese direkte Adressierungsmöglichkeit von PL/M bewirkt den geringen Grad der Wartbarkeit und Transportabilität. Diese Art der Adressierung verursacht trotz gewisser Vorteile bei der Entwicklung später bei der Wartung große Schwierigkeiten, wenn sich die Länge des Programms ändert und es später notwendig wird, neue Speicheradressen festzulegen.

Eine weitere Praxis bei höheren Programmiersprachen mit ähnlichen Auswirkungen ist der Gebrauch von Pointer-(indirekten) Variablen, um absolute Speicheradressen festzulegen. Diese Art der Programmierung führt zu den gleichen Schwierigkeiten beim Manipulieren der Speicheradressen. Obwohl es sich um eine schlechte Programmiertechnik handelt, ist sie bei PL/M und Mikroprozessor-PASCAL erlaubt. Die Syntax von Mikroprozessor-PASCAL zwingt den Programmierer jedoch, Statements im Programm zu machen, die klarstellen, was durchgeführt wird, außerdem können die Speicherbereiche, für die diese Pointer erlaubt sind, beschränkt werden. Bei PL/M sind solche Statements nicht von normalen Zuweisungs-Statements zu unterscheiden.

Eine gute Programmiersprache sollte so konzipiert sein, daß Daten flexibel verwendet werden können (Punkt 10). Damit die Ausdrücke klar sind, sollten sich die Datentypen über reelle Zahlen, ganze Dezimalzahlen, Boolesche Variable, Textzeichen, Felder, Sets, Records und Kombinationen aus diesen Elementen erstrecken. PASCAL erlaubt es dem Programmierer, seine eigenen Datentypen festzulegen, was die Zuverlässigkeit in hohem Maße verbessert und die Zahl von Programmierfehlern reduziert, die üblicherweise bei der Behandlung von Daten auftreten. Mit PASCAL kann man Problemlösungen durch vertraute Terminologie beschreiben, z. B. kann eine Variable „Wochentag" vom Typ „MONTAG", „DIENSTAG" usw. sein. Selbst COBOL besitzt eine solche Möglichkeit nicht.

Die Fähigkeit des Compilers, Fehler- und Verträglichkeitsprüfungen vorzunehmen, ist nicht ausschließlich eine Funktion des Compileraufbaus, sondern auch abhängig von der Programmiersprache. Softwareentwicklung ist einfacher, wenn soviel Fehler wie möglich während der Compilierung erkannt werden. Dies ist um so wichtiger, je größer das Programm wird. Eine der wichtigsten Eigenschaften von PASCAL ist die Möglichkeit, Unverträglichkeit von Datentypen bei der Datenmanipulation zu lokalisieren. Implizite Mixed-Mode-Operationen an Daten sind verboten.

PASCAL steuert den Zugriff auf Daten (Punkt 12) in einer Weise, die den Programmierer daran hindert, über Prozeduraufrufe unverträgliche Manipulationen an Daten auszuführen. Solche Operationen können

außerordentlich überraschende Fehler in einem großen System verursachen, besonders, wenn der Fehler datenabhängig ist.

Run-Time-Support

Die meisten höheren Programmiersprachen, einschließlich PASCAL können auf eine der zwei Arten ausgeführt werden:

1. Die Statements der höheren Programmiersprache werden während des Programmablaufs in den Maschinencode übersetzt (interpretiert).
2. Die Statements der höheren Programmiersprache werden vor dem Programmablauf übersetzt (compiliert).

Beim Mikroprozessor-PASCAL-System kann der Anwender beide Methoden benutzen. Compilierter Code ist bei der Programmausführung schneller, aber es wird ein größerer Speicherbereich bei umfangreichen Applikationen erforderlich. Zunächst benötigt man weniger Speicher beim compilierten Code, weil ja kein Interpreter vorhanden ist. Trotzdem braucht das Anwenderprogramm mehr Platz, weil der Maschinencode weniger kompakt ist als der interpretierbare Code.

Um die beste Transportabilität zu erreichen, bietet TI einen Softwarekern passend zur Hardware. Dieser nur wenig Speicherplatz belegende Bereich enthält neben anderen Dingen eine Verwaltung für den „Stack", in dem die Benutzerdaten und Systemvariablen enthalten sind. Dieser Bereich ist nicht transportabel, liefert aber eine Basis für PASCAL, auf der die Executiv-Software sowie die Anwendersoftware aufbaut. Auf diese Weise wird sowohl das Betriebssystem als auch das Anwenderprogramm maschinenunabhängig. Um das System auf eine Hardware mit anderem Instruktionssatz zu transportieren, wird eine relativ einfache Modifikation des Mikroprozessor-PASCAL-Codegenerators erforderlich. Der Compiler selbst, der den interpretierbaren Zwischencode (P-Code) erzeugt, braucht nicht verändert zu werden.

Wenn komplexe Probleme im Echtzeitbetrieb gelöst werden sollen, ist es üblich, die Lösung in verschiedene gleichzeitig ablaufende Subtasks aufzuteilen. Umfangreiche zeitabhängige logische Abläufe werden in kleinere und leicht zu handhabende parallele Steuerschleifen aufgelöst. Es ist fast unmöglich, dies in einem sequentiellen logischen Flußdiagramm auszudrücken, da die unterschiedlichsten Zustandskombinationen, die in Anwendung auftreten, dargestellt werden müßten.

Um die parallelen Programmabläufe zu verwalten, muß ein übergeordnetes Programm die zur Verfügung stehende Hardware verteilen und den gleichzeitigen Ablauf nach vorher aufgestellten Regeln planen. Dieses Management, das auch „*Concurrent-Processing*" genannt wird, ist ein Teil des Betriebssystems und wird in der Mikroprozessor-PASCAL-Library, sowohl für interpretierbaren (MPIX), als auch für Maschinencode (MPX),geliefert. Die Programm-Teile, die parallel ausgeführt werden, werden mit Prozeß bezeichnet. Jeder Prozeß besitzt seine Priorität bezüglich der anderen Prozesse, eine Aufzeichnung seiner Aktivitäten, Daten und Prozedur-Subroutinen. Die Aufgabe des MPX/MPIX ist die Verwaltung dieser Prozesse auf verschiedenen Ebenen der erforderlichen Priorität. Die höchste Priorität sind Interrupts.

Die Prozesse müssen synchronisiert werden und miteinander kommunizieren. In der einfachsten Form geschieht das durch Semaphores. Ein Semaphore wird durch Systemaufrufe verfügbar. Auf der nächsten Stufe der Komplexität werden die Semaphores in einem „Briefkasten"-Kommunikations-Schema benutzt, mit Nachrichten, die zwischen den Prozessen versandt und empfangen werden. Das geschieht mit Semaphores, die signalisieren, daß eine Nachricht vorliegt.

Die allgemeinste Kommunikation zwischen den Prozessen geschieht über Files, die über die PASCAL-E/A-Möglichkeiten angesprochen werden. Ein Prozeß führt PASCAL-WRITE aus, während der andere READ ausführt. Auch hier werden wieder Semaphores benutzt, um die Aktivitäten zweier oder mehrerer Prozesse zu synchronisieren, die auf ein File Zugriff haben.

Nützliche Dienstprogramme

Neben Compiler, Interpreter und der umfangreichen Sammlung von Betriebssystem-Moduln, die Concurrent-Programming unterstützen, liefert TI in seinem Mikroprozessor-PASCAL-System einen vollständigen Satz von Dienstprogrammen.

Das beginnt beim bildschirmorientierten Editor mit eingebautem PASCAL-Syntax-Check. Damit kann Programmtext nicht nur sehr komfortabel und rasch bearbeitet werden, sondern es ist bereits während der Eingabe eine Überprüfung auf syntaktische Richtigkeit des PASCAL-Programms möglich, ohne daß vorher ein Compilerlauf nötig ist.

Eine umfangreiche Sammlung von Debug-Routinen erlaubt das Austesten der PASCAL-Programme auf zwei Ebenen: Auf dem Mikroprozessor-Entwicklungssystem AMPL ist ein reiner Software-Funktionstest möglich, wobei Interrupts und Hardware E/A simuliert werden können. Ist die Hardware für das endgültige Anwendersystem fertiggestellt, so kann das ordnungsgemäße Zusammenspiel von Hard- und Software in Echtzeit unter Zuhilfenahme des AMPL-Emulators ausgetestet werden. Gleich auf welcher Ebene man austestet, der Debugger unterstützt neben Breakpoint, Single-Step, Trace auf Statement- oder Prozeß-Ebene, Stack- und Heap-Anzeige/Modifikation auch Befehle speziell zum Austesten von parallel laufenden Prozessen. Der Zugriff erfolgt dabei symbolisch, d. h. direkt auf PASCAL-Statements, weshalb man von „Source-Debug" spricht.

Kenneth Wickham ist Manager für Mikroprozessor-Software bei Texas Instruments in den USA. Er erlangte den BSEE an der Northwestern University. Er war vorher Leiter in der Semiconductor's Management Systems Group bei Texas Instruments. Seine früheren Spezialgebiete bei TI waren Echtzeit-Prozeßsteuerung, Betriebsmanagement-Informationssystem und Simulation von diskreten Ereignissen.
Diese Arbeit wurde mit der freundlichen Genehmigung der Redaktion aus IEEE Spectrum 1979, H. 3, S. 35...41 übernommen. Aus dem Englischen übertragen von *P. von Bechen*, ergänzt und überarbeitet von *M. Rudyk*, TI Deutschland.

Die „Fallgruben" von PASCAL

Programmiersprachen können nicht beurteilt werden, ohne mit Programmierern zu sprechen, weil Software ein Werkzeug zur Problemlösung ist. Und Problemlösung ist immer noch eine subjektive Fähigkeit, nicht aber eine Wissenschaft.

Was denken Ingenieure und Programmierer, die mit Echtzeit-Test- und -Meßgeräten zu tun haben, über PASCAL? Um das herauszufinden, wurde eine Gruppe interviewt, die täglich diese Sprache benutzen. Die Gesprächsrunde wurde bei Tektronix in Beaverton, Ore., abgehalten. Wie viele andere Firmen, die mit hochstehender Technologie arbeiten, befaßt man sich bei Tektronix schon seit Jahren mit PASCAL. Verschiedene Versionen dieser Sprache sind für die Computer von Tektronix verfügbar.

Bei der Diskussion kamen die Grenzen von PASCAL zur Sprache. Der Gedanke dabei war nicht, potentielle Anwender abzuschrecken (die meisten Beteiligten am Gespräch sind enthusiastische Befürworter von PASCAL), sondern es sollten einige „Fallgruben" beleuchtet werden, über die noch nicht näher berichtet worden ist.

Die Nachteile scheinen sich in diese Kategorien einteilen zu lassen:

- Die Probleme eines Ingenieurs, der über Erfahrungen mit anderen Sprachen wie FORTRAN oder BASIC verfügt und der nun zu PASCAL übergehen möchte,
- „Programmier-Flaschenhälse" – Es handelt sich hierbei um Eigenheiten in der Syntax der Sprache, die bestimmte Aufgaben schwieriger machen, als sie sind,
- Beschränkungen von PASCAL als ein Werkzeug zur Erzeugung von Maschinencode für Mikroprozessor-Systeme.

Nie mehr „Spaghetticode"

Spaghetticode ist eine Bezeichnung für eine verwikkelte Aneinanderreihung aus schwer verständlichem Code ohne erkennbaren Anfangs- oder Endpunkt. Spaghetticode kann in jeder Sprache geschrieben werden, einschließlich PASCAL (*Bild 2a*). Aber manche Sprachen, so z. B. PASCAL, haben eine Syntax, die eine klare Ausdrucksweise ermöglicht und fördert.

Bei Tektronix haben sich die Ingenieure durch Selbststudium oder durch Lehrgänge mit Firmenunterstützung auf PASCAL eingearbeitet. Zwei Teilnehmer der Gesprächsrunde, *Stephen Heitman*, Software-Ingenieur, und *Robert Reed*, Software/Hardware-Ingenieur, führen solche Kurse durch. Eine Sache, auf die sie besondere Betonung legen, ist die Notwendigkeit, den Hang der Studenten zum Spaghetticode, den sie von BASIC und FORTRAN gewöhnt waren, zu bekämpfen.

Ein weiterer Gesprächspartner war *Paul Blattner*, ein Software-Forschungsingenieur, *Richard Kaufmann*, Software-Ingenieur, *Stephen Dum*, Forschungsmanager für neue Logikbausteine, *Paul Hoefling*, wissenschaftlicher Software-Ingenieur, und *Lynn Saunders*, Hardware-Ingenieur. Die wichtigsten Kernsätze der Diskussion:

Reed: „Das größte Problem, das ich beim Ausbilden einer Gruppe von Ingenieuren habe, ist, daß die meisten mit BASIC groß geworden sind und glauben, daß dies die beste Sprache sei. Sie behalten diese Meinung bei, während sie PASCAL lernen. Ich neige dazu, zu übertreiben und mache BASIC schlecht, wo es geht. Deswegen sind sie mir manchmal ein bißchen böse. Es ist ein Kreislauf, aus dem man schlecht herauskommt."

Blattner: „Es ist ja nicht nur eine andere Art der Programmierpraxis. Es ist auch eine andere Art des Denkens. Wenn man von BASIC auf PASCAL übergeht, wird man gezwungen, von der ‚Spaghetticode-Denkweise' zur ‚Top-down-Methode' überzugehen. Das ist ein schwieriger Prozeß des Umdenkens."

Reed: „Das stimmt. Das sind aber nur die allgemeinen Gesichtspunkte des Problems."

Anscheinend erfordert die strukturierte Programmierung bei PASCAL neue Einsichten. Mr. Blattner beschreibt, wie es war, als ihm „das Licht aufging".

Blattner: „Ich erinnere mich, daß, nachdem ich strukturierte Methoden gelernt hatte und versuchte, dies beim Programmieren anzuwenden, ich auch ein wenig in dieser Art codierte – mit großen Schwierigkeiten – bis ich das Wesentliche erkannt habe. Ich hatte es eigentlich bereits aufgegeben, weil es ohne die Einsicht, daß es vorteilhaft ist, für mich lächerlich aussah, in dieser Art zu codieren. Aber irgendwie bin ich dann darauf gekommen und alles ist jetzt ganz natürlich."

Kaufmann: „Es ist so, als ob man zu einer Bowling-Ausbildung geht, nachdem man diesen Sport schon 20 Jahre betreibt. Man hat sich in dieser Zeit einiges angewöhnt, das man nun vergessen soll. Dafür gibt es andere Dinge, die man von Grund auf neu lernen muß."

Heitman: „Man hat ein Gesamtkonzept und dieses wird in immer feinere Untergruppen aufgeteilt. Diese Vorgehensweise ist besser als von Detailpunkten auszugehen, ihre Beziehungen und Verknüpfungen untereinander herauszufinden und alles zu einem Gesamtkonzept zusammenzufassen. Anstatt daß man eine Lösung beim Lösen des Problems sucht, geht man den umgekehrten Weg."

Einstimmiges Lob für PASCAL

Alle Software-Ingenieure bei Tektronix lobten die Nützlichkeit und Flexibilität dieser Sprache.

Reed: „Ich finde die Einfachheit der Verzweigungsstrukturen bei PASCAL besonders gut. Es gibt sehr viele einfache Verzweigungsstrukturen, die man kombinieren kann um so viele Aufgaben auszuführen, wie man möchte. Aber das wichtigste bei PASCAL ist die Fähigkeit zur Datenstrukturierung – weil das Zentralpunkt der ingenieurmäßigen Programmierung ist, nämlich das Entwikkeln von Datenstrukturen und deren Manipulation, um die gewünschten Ergebnisse zu erhalten."

Blattner: „Von allen Sprachen, mit denen ich gearbeitet habe – und ich habe mit FORTRAN, COBOL, BASIC, PL/1 und nun PASCAL gearbeitet –, scheint mir PASCAL die leistungsfähigste Sprache in dieser Größenklasse zu sein. In anderen Worten, man hat eine Sprache, die so knapp gehalten ist wie viele der neueren Sprachen, die bei Mikroprozessoren Verwendung finden, aber sie scheint so leistungsfähig, wenn nicht leistungsfähiger zu sein, wie man das von den umfangreichen Compilern der traditionellen Sprachen gewöhnt ist.

Kürzlich habe ich ein Suchprogramm ausgearbeitet und mußte etwas haben, was neben richtig oder falsch liegt – ich brauchte eine vierwertige Logik. Daher habe ich einfach einen vierwertigen Typ definiert und nun kann ich Werte – diese vier Werte – einer Variablen zuweisen.

Mehr als andere Sprachen, die ich kenne, veranlaßt einen PASCAL, in strukturiertem Code zu schreiben. Ich arbeite jetzt schon vier Monate mit PASCAL und habe tatsächlich noch kein ‚GO-TO' oder ein Label benutzt...

In anderen Worten: PASCAL führt zum Top-Down-Design und durch seine Struktur zum strukturierten Code."

Aber bis zu einem gewissen Grade kann jede Sprache blockstrukturiert sein. Mr. Reed berichtet von seinen Erfahrungen mit FORTRAN.

Reed: „Ich habe eine Menge in FORTRAN programmiert. Ich glaube, daß der Code, den ich in FORTRAN schreibe, blockstrukturiert ist und sich an die Top-Down-Architektur anlehnt, obwohl derartige Konstruktionen in der Standardsprache nicht verfügbar sind. Ich mache das mit meinen zusätzlichen Kommentarstrukturen, indem ich von der wirklichen Struktur des Codes abrücke. Ich schreibe die Routinen und Subroutinen so, daß sie von oben nach unten ablaufen und ich vermeide möglichst Programmsprünge. Ich benutze die GO-TO-Statements und ähnliche nichtstrukturierte Steuerfunktionen, um die Blockstruktur-Techniken zu implementieren. Außerdem benutze ich eine Menge CONTINUE-Statements."

PASCAL hat auch Schwächen

Es gibt mehrere Schwächen bei PASCAL, aber nach Aussage von *Mr. Dum* soll es bald eine vernünftige Übereinkunft zwischen den PASCAL-Usergroups über Erweiterungen zur Behebung der Mängel geben. Unabhängig von den Erweiterungen gibt es tatsächlich unerfreuliche Aspekte dieser Sprache.

Dum: „Ich glaube, einer der wirklichen Mängel des gegenwärtigen PASCAL ist, daß man einen Code schreiben muß, der jeder Konstanten einen Wert zuordnet. Das läuft darauf hinaus, daß man zwei- bis dreimal soviel Platz in seinem Programm braucht, um die Aufgabe zu lösen... Man muß der Aneinanderreihung von Konstanten Platz zuweisen, dann jedes Element nehmen und verschieben."

Reed: „PASCAL ist entwickelt worden, um jemandem Programmieren beizubringen. Deshalb gibt es sehr strenge Datendefinitionen. Das spart sehr viel Aufwand bei der Fehlersuche, aber manchmal – wenn man z. B. versucht, bestimmte anspruchsvollere Datenstruktur-Manipulationen vorzunehmen – wirkt sich das behindernd aus.

Eine andere Schwäche von PASCAL, zumindest einiger Versionen, ist die fehlende Zuweisung, um Felder mit variabler Länge für String-Operationen zu erlauben. Noch ein weiterer Nachteil ist die mangelnde Modularität zwischen verschiedenen Teilen eines Projekts, damit man nicht für jedes Modul alle externen Referenzen bis ins kleinste Teil definieren muß. Ich glaube auch, daß es schwierig ist, Prozesse parallel ablaufen zu lassen, wenn man nicht gerade Concurrent PASCAL benutzt."

Hoefling: „Einer der Mängel bei Standard-PASCAL ist das Fehlen des ELSE bei CASE-Statements. Wenn die CASE-Abfrage auf keines der CASE-Labels paßt, ergibt sich bei PASCAL ein undefinierter Zustand."

Heitman: „Ein anderes Problem ist das Fehlen von Exponential-Operatoren. Das ist absurd. Ich beschäftige mich vorwiegend mit numerischer Analysis. Ich finde es wirklich ärgerlich, keinen Exponential-Operator zu haben. Wenn ich gewisse mathematische Routinen auszuführen habe, muß ich spezielle Prozeduren schreiben, um dies tun zu können."

Saunders: „Es ist schwierig, PASCAL und Assembler-Code zu mischen. Schnelle Programmausführung ist lebenswichtig für Echtzeitanwendungen wie z. B. einem Multimeter. Der Entwickler der Firmware sucht ein Werkzeug, das ihm einerseits Assembler-Code liefert, aber andererseits das Format einer höheren Program-

A

```
(* MAIN PROGRAM *)
BEGIN
   RESET (input files);
   REWRITE (output files);
10: IF EOF (employees) THEN GOTO 99
   READ (employee info and hours worked);
   IF NOT employee is exempt THEN GOTO 20
   STRAIGHTTIME (40);
   BENEFITS (40);
   GOTO 50;
20: IF more than 40 hours worked THEN GOTO 30
   STRAIGHTTIME (hours worked);
   GOTO 40;
30: STRAIGHTTIME (40);
   OVERTIME (hours worked—40);
40: BENEFITS (hours worked);
50: WRITECHECK;
   UPDATEFILE;
   GOTO 10;
99: PRINTTOTALS;
END. (* PROGRAM *)
```

B

```
(* MAIN PROGRAM *)
BEGIN
   RESET (input files);
   REWRITE (output files);
   WHILE NOT EOF (employees) DO
      BEGIN
      READ (employee info and hours worked);
      IF employee is except THEN
         STRAIGHTTIME (40);
         BENEFITS (40)
         END
      ELSE
         IF more than 40 hours worked THEN
            BEGIN
            STRAIGHTTIME (40);
            OVERTIME (hours worked—40);
            BENEFITS (hours worked)
            END
         ELSE
            STRAIGHTTIME (hours worked);
            BENEFITS (hours worked)
            END;
      WRITECHECK;
UPDATEFILE;
END; (* EMPLOYEE LOOP *)
   PRINTTOTALS;
END. (* PROGRAM *)
```

Bild 2. „Spaghetticode" kann unglücklicherweise in jeder Sprache geschrieben werden, einschließlich PASCAL, wie es in dem oberen Beispiel (A) dargestellt ist. Das gleiche ist in Blockform (B) ausgeführt. Programm A ist schwerer zu lesen, weil es GO-TO-Statements und Lables enthält. Deswegen benützt das Programm B Steuerworte (WHILE, BEGIN, IF-ELSE, END) um zu klären, wie der Algorithmus funktioniert. Bei den üblichen Subroutinen, die auch eine Art strukturiertes Statement sind, wäre A viel länger als B und auch schwerer zu verstehen. Das Programm stammt von *Paul Blattner*

miersprache besitzt, um die Stacks und Register in den Griff zu bekommen.

Der Entwickler kann 80 % seiner Arbeit mit der höheren Programmiersprache wie z. B. PASCAL erledigen, aber dann bleiben noch die 20 % mühevolle Kleinarbeit, um dem Multimeter die höchste Geschwindigkeit zu geben. Das erfordert Erweiterungen der höheren Programmiersprache, um direkten Zugriff im Assemblercode auf Hardwareadressen zu haben.

Die meisten Leute, die Erweiterungen für PASCAL verlangen, denken, sie könnten dieses Problem durch Moduln lösen, die für sich compilierbar sind. Ich meine, daß die Aufteilung eine Stufe höher vorgenommen werden sollte. Ich würde PASCAL als eine Sprache sehen, die man zur Entwicklung von Firmware benutzen kann, mit einer Art Schalter, der es einem erlaubt, den Compiliervorgang an einem bestimmten Punkt des Quellencodes zu stoppen, indem man sagt: ‚OK – Compiler, du hast deine Arbeit getan. Weil es jetzt schneller gehen muß, wird jetzt die Codierung in Assembler weitergeführt'."

Kaufmann: „In automatischen Testsystmen steht es um dieses ‚Schalterproblem' besonders schlecht, weil es keinen Weg gibt, dem PASCAL-Compiler zu sagen: ‚Ich bin jetzt bei der zwölften verschachtelten Programmprozedur angelangt und möchte anhalten, weil ich bemerkt habe, daß irgendein Chip nicht richtig läuft'."

Heitman: „Es ist schwierig, eine ‚Bit-Verarbeitung' mit PASCAL durchzuführen... Bei bestimmten praktischen Programmieranwendungen, wie z. B. beim Maskieren, ist es so gut wie unmöglich, effiziente Programme in Standard-PASCAL zu schreiben. Es wäre natürlich schön, wenn man in der Lage wäre, eine Art Assemblercode in das Programm zu integrieren. Man kann es mit Assemblercode oder FORTRAN-Programmen verknüpfen im Tek-Cyber-Programm, jedoch nicht mit Standard-PASCAL."

Wie TI PASCAL verbesserte

Nachdem TI mehrere Jahre Erfahrung durch internen Gebrauch von PASCAL gewonnen hatte, wurden etwa ein halbes Dutzend Änderungen in der Syntax und im Compiler dieser Sprache vorgenommen, um aus PASCAL ein „lebenstüchtiges" Werkzeug zu machen.

Unter den Syntaxveränderungen sind flexiblere E/A-Aufrufe, zuverlässigere Definitionen der Statements WITH, FOR und CASE, eine konventionellere Definition der Booleschen Operatoren (wie bei FORTRAN) und eine Restriktion des Variablen-Bereichs einer Funktion, damit die Variable nicht versehentlich verändert wird, wenn sie beim Programmablauf aufgerufen wird.

Im einzelnen wurden die E/A-Befehle GET und PUT durch die Standard-Funktionen READ und WRITE ersetzt, die auf Text- oder Nichttext-Files erweitert wurden. Es stimmt, daß die ursprünglichen E/A-Statements den Transport leichter vornehmen, aber das wurde hier geopfert, um größere Flexibilität zu erlangen.

Erweiterte Integer-Typen (LONG INT) werden über einen Bereich von $-2^{31}...2^{32}-1$ verarbeitet.

PASCAL hat ein besonders zeitsparendes WITH-Statement, das für ein einzelnes Record-Statement in einer DO-Schleife verwendet werden kann. Im WITH-Statement ist es nicht notwendig, die Identifier der Variablenfelder dieses Records anzugeben. Mikroprozessor-PASCAL erlaubt den Gebrauch von expliziten Mehrfach-WITH-Statements ohne Doppeldeutigkeit.

Im FOR-Statement, mit dem Schleifen in einer festgelegten Anzahl durchlaufen werden, muß die Laufvariable nicht erklärt werden und ist nur innerhalb der Schleife bekannt. Das erhöht die Zuverlässigkeit des FOR-Statements. In der Originalform von PASCAL hätte die Variable außerhalb der Schleife in einem anderen Zusammenhang verwendet werden dürfen. So hätte nach dem Verlassen der Schleife diese Variable leicht unbeabsichtigt verändert werden können.

PASCAL verfügt über ein CASE-Statement, das eine Befehlsfolge unmittelbar nach einer einfachen Überprüfung eines Wertes, vor der das Wort CASE steht, ausführt. Mehrere CASE-Labels können benutzt werden. Schwierigkeiten können entstehen, wenn der Compiler ein CASE-Label sucht, welches nicht existiert. TIP erlaubt eine OTHERWISE-Bestimmung für den Fall, daß ein CASE-Label fehlt. Es erlaubt auch Subrange-Labels bei CASE um Werte in einem abgegrenzten numerischen Bereich festzulegen. Die Verbesserungen des 9900-Compilers umfassen die folgenden Punkte:

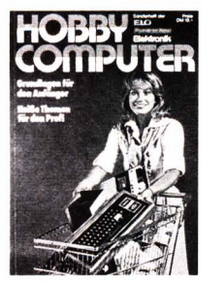
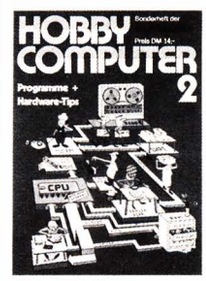
- Er unterstützt Random-Access-Files. Ein Random-File ist implementiert als Satz von logisch geordneten Records, beginnend bei Null. Auf die Records wird direkt zugegriffen durch die relative Position im File.
- Er läßt verschiedene Routinen die gleiche allgemeine Globalvariable benutzen ohne Mehrfachzuweisungen an den Run-Time-Stack. Das ermöglicht den gleichen Wert der Variablen beizubehalten von einem Aufruf der Routine zum nächsten.
- Er erlaubt das Einbinden von Run-Time-Prüfpunkten, um z. B. Feldgrenzen, Pointer, Satz und Subranges auf Gültigkeit zu prüfen.
- Er erlaubt einen ESCAPE-Mechanismus, um die Ausführung in Abhängigkeit von Fehlerbedingungen oder externen Ereignissen zu beenden.
- Er bietet einen expliziten TYPE-OVERRIDE-Operator, damit der Compiler aus der Typ-Überprüfung einer Routine herauskommt. Er bietet ein ASSERT-Statement, das die Einfügung von benutzerdefinierten Run-Time-Prüfpunkten in das Programm erlaubt.
- Er ermöglicht getrennte Compilation von Prozeduren und das Einbinden von Assembler-Routinen. Der Compiler überprüft zusätzlich zur normalen Typ-Überprüfung, wie die Daten zwischen Routinen ausgetauscht werden.
- Er bietet E/A-Möglichkeiten zur Bitmanipulation, dadurch, daß der PASCAL-Code für die direkte Adressierung der Hardware-Kommunikationsregister-Einheit (CRU) des Mikroprozessors TMS 9900 erweitert wird. Dieses Merkmal bindet, wenn es benutzt wird, den Compiler an spezifische Hardware und unterbindet damit die Transportabilität, aber Bitverarbeitung ist so wichtig für die Laufzeitoptimierung, daß der Nachteil gerechtfertigt ist.

Franzis-Sonderhefte

für alle Mikrocomputer-Interessierte

Welchen Weg der Mikrocomputer gehen wird, läßt sich noch nicht vorherbestimmen. Vor allem im Anwendungsbereich steht die Entwicklung erst am Anfang. Eines aber ist absehbar, jeder der auch nur am Rande mit Elektronik zu tun hat, wird davon berührt werden bzw. ist heute schon fest damit verbunden.

Mit diesen Sonderheften wollen wir jeden Berufs- oder Hobby-Elektroniker aktuell über den neuesten Stand informieren. Welches Heft für Sie als Informationsquelle in Frage kommt, können Sie aus nachstehender Tabelle ersehen.

FUNKSCHAU-Sonderheft: Programme für Kleincomputer und Taschenrechner	ELO-Sonderheft: Dem Mikrocomputer auf's Bit geschaut	FUNKSCHAU-Sonderheft Mikrocomputer – eine Einführung in kleinen Schritten	Sonderheft der ELO, FUNKSCHAU und ELEKTRONIK Hobbycomputer 1 	FUNKSCHAU-Sonderheft Hobbycomputer 2 	ELEKTRONIK-Sonderheft 3 Mikroprozessoren
Für alle Computerbesitzer und solche, die es werden wollen	Anfänger	Anfänger	Anfänger und Fortgeschrittene	Fortgeschrittene	Industrielle Anwender
Programme für Taschenrechner (HP- und TI-Serien), für Basic-Computer und für die Mikroprozessoren Z-80 und 6502 auch in Maschinensprache Zahlreiche neue Beiträge.	Hardwareorientierte Einführung mit µP 2650. Sammlung der in der ELO bereits erschienenen gleichlautenden Beitragsreihe.	Hard- und Software-Grundlagen. Gesammelte Beiträge der in der FUNKSCHAU bereits erschienenen gleichnamigen Reihe.	Grundlagen, einfache Basic- und Maschinenprogramme. Überwiegend Beiträge, die bisher nirgendwo erschienen sind.	Basic- und Maschinenprogramme, Hardware-Tips. Ausschließlich exklusive Beiträge.	Programmierhilfen, Markttendenzen, 16-Bit-Prozessoren, Adressen von Herstellern. Überwiegend Beiträge aus der ELEKTRONIK.
80 Seiten 14.80 DM	144 Seiten 18.– DM	68 Seiten 9.50 DM	136 Seiten 19.– DM	80 Seiten 14.– DM	136 Seiten 18.– DM

Wo gibt es diese Sonderhefte:

Sie erhalten diese Hefte bei allen Bahnhofsbuchhandlungen, Elektronik-Bauteilehändlern, größeren Buchhandlungen oder direkt beim Franzis-Verlag.

Bitte haben Sie Verständnis, daß der Verlag Einzelhefte aus organisatorischen Gründen nur gegen Vorauszahlung liefern kann. Wir bitten Sie, in diesem Fall als Bestellung den genannten Betrag plus 1.50 DM Porto auf unser Postscheckkonto München Nr. 813 75-809 mit genauer Nennung des jeweiligen Titels zu überweisen. Bitte vergessen Sie nicht, auf dem Zahlungsbeleg in Druckschrift Ihre volle Anschrift anzugeben. Sofort nach Eingang der Zahlung senden wir Ihnen das Heft zu.

Franzis-Verlag

Karlstraße 37, 8000 München 2
Telefon (0 89) 51 17-2 39

Die Hefte erhalten Sie auch in der Schweiz beim Verlag Thali AG, CH-6285 Hitzkirch

und in Österreich beim Fachbuch Center Erb, A-1061 Wien, Amerlingstraße 1.

Dipl.-Ing. Eberhard Enger

Programmieren mit PASCAL

Auch in Mikroprozessorsystemen sind mittlerweile höhere Programmiersprachen anwendbar geworden. Dabei hat sich PASCAL als praktische und universelle Anwendersprache herausgestellt, die durch ihre Kontrollanweisungen und Datenstrukturen das systematische Programmieren unterstützt. Hier soll anhand von praktischen Beispielen das Programmieren mit PASCAL vorgestellt werden.

1 Bedeutung von PASCAL und derzeitiger Stand

Mit dem Erfolg von PASCAL ist erneut eine heftige Diskussion über die „beste" Programmiersprache entbrannt. Als ein Indiz für die nun schon breitere Verwendung von PASCAL mag gelten, daß die Normung auf nationaler (DIN) und internationaler Ebene (ISO/TC 97/SC5) betrieben wird. Sowohl FORTRAN (trotz seines reichhaltigen Bibliothekangebots) als auch BASIC (die beliebteste Sprache auf kleinen Computern) haben eine ernstzunehmende Konkurrenz bekommen. ALGOL-60 und natürlich auch das kompliziertere ALGOL-68 erscheinen auf dem Bereich des praktischen Programmierens schon überrundet, was nicht verwundert, da PASCAL über ALGOL hinausgeht, aber leichter zu erlernen ist. Für die Verbreitung von PASCAL spricht, daß die beiden Entwicklungsziele von *Niklaus Wirth*, nämlich leichte Erlernbarkeit und Lesbarkeit für den Benutzer sowie leistungsfähige Übersetzung durch den Computer, überzeugend realisiert wurden. Außer diesen beiden Zielen haben sich noch folgende Punkte als relevant erwiesen: PASCAL soll allgemein verwendbar sein – was natürlich nicht heißt, daß es für alles benutzt werden kann. Die noch immer scharfe Trennung zwischen den einerseits wissenschaftlich-technischen und andererseits mehr kaufmännisch orientierten Programmiersprachen (FORTRAN, COBOL) soll aufgehoben, d.h. die Behandlung numerischer und nicht numerischer Probleme mit einer Sprache ermöglicht werden. Zu diesem Punkt gehört auch, daß vorhandene und erprobte Software weiterhin benützt werden kann.

Ein wichtiges Kriterium für die Qualität einer Sprache ist der Gesichtspunkt der Austauschbarkeit von Programmen zwischen verschiedenen Rechnern (Stichwort: Übertragbarkeit, *portability*). Sind Programme ohne Änderungen – das Ideal – zu übertragen, so wird teure Mehrfachentwicklung vermieden und der Zugang auf Software aus anderen Quellen erleichtert. Notwendig dazu ist eine eindeutige Formulierung eines Sprach-Standards *und*: daß sich die Compiler nach diesem Standard richten. Da bei der Realisierung eines Compilers auf einem konkreten Computer maschinenabhängige Faktoren eine Rolle spielen, führte dies im gewissen Maße auch bei PASCAL dazu, daß verschiedene „Dialekte" entstanden sind. Der derzeitige Entwurf zur internationalen Standardisierung basiert daher sehr eng auf dem Report von *Jensen/Wirth* |1| und versucht so, einem Wildwuchs entgegenzuwirken. Es besteht daher berechtigte Hoffnung, daß PASCAL eine weitgehend herstellerunabhängige Sprache bleibt.

Manche Beschränkungen – z. B. keine dynamische Felder wie in ALGOL-60, keine festvereinbarten Variablen (A, A1, A$) wie in BASIC, das Fehlen der doppelten Genauigkeit wie bei FORTRAN und COBOL – werden von vielen Programmierern nur schwer akzeptiert, was zunächst eine Ablehnung von PASCAL hervorruft. Erfahrungen zeigen aber, daß nach einer Umstellungszeit diese Punkte an Bedeutung verlieren. Über die Schwächen von PASCAL wird noch zu gegebener Zeit zu reden sein.

Die Anwendungstiefe von PASCAL läßt sich u. a. daran demonstrieren, daß die Verbreitung der Sprache wesentlich durch den sogenannten PASCAL-P-Compiler gefördert wurde. Dieser Compiler ist in PASCAL selbst – also in der Sprache, die er übersetzt – geschrieben (Länge der Programm-Liste: ca. 3600 Zeilen). Er erzeugt aus einem (korrekten) Programm einen

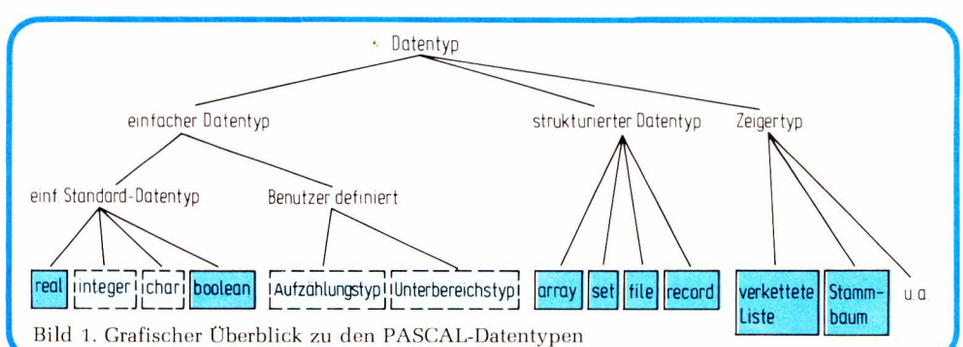


Bild 1. Grafischer Überblick zu den PASCAL-Datentypen

allgemeinen *P-Code*, also keinen speziellen Assemblercode. Dieser P-Code ist für einen hypothetischen Computer mit einem Stapel (*stack computer*) definiert und enthält in etwa die von Assemblern her bekannten Befehle. Für die Anpassung (Implementierung) an eine bestimmte Rechenanlage ist zunächst ein kleiner Interpreter oder Assembler zur Übersetzung des P-Codes des PASCAL-P-Compilers in den Maschinencode erforderlich. Darauf aufbauend kommt man nach einigen Optimierungsschritten zu einem leistungsfähigen PASCAL-Compiler. Auf diesen Aspekt wird im Abschnitt 8 nochmals eingegangen. Da ein solch umfangreiches Programm auch von einem weniger geübten Programmierer gelesen und nach einer Einarbeitungszeit verstanden werden kann, bietet es die Möglichkeit, sich Problemlösungen für das Übersetzen anzueignen und auf eigene Aufgabenstellungen (z. B. Entwicklung eines Cross-Assemblers für einen Mikroprozessor) anzuwenden.

Nach der ursprünglichen Beschreibung der Programmiersprache PASCAL von *N. Wirth* im Jahre 1971 [2] erfolgte einige Jahre später eine Revision [3, 4, 1]. Der Name PASCAL wurde von *N. Wirth* zu Ehren des französischen Philosophen und Mathematikers *Blaise Pascal* (1623...1662) gewählt, der eine der ersten funktionstüchtigen Rechenmaschinen gebaut hatte. Das „User Manual and Report" von *Jensen/Wirth* [1] ist z. Zt. das anerkannte Standardwerk über die Programmiersprache PASCAL. Die in Aussicht genommene Standardisierung weicht nur in wenigen Punkten von dieser Fassung ab. Der deutsche Bericht [3] enthält ausschließlich den übersetzten Report aus [1] und ist somit als Lehrbuch weniger geeignet. Die mittlerweile in deutscher Sprache [5, 6, 7, 8, 9] erhältlichen Pascal-Lehrbücher sollen hier zunächst erwähnt und später kurz besprochen werden.

Mitte der siebziger Jahre wurde die „Pascal User's Group" (PUG) gegründet, um die Verbreitung von PASCAL zu fördern. In ihren Mitteilungen – den „Pascal News" – finden sich Diskussionen über die Sprache, in PASCAL formulierte Algorithmen (z. B. Schönschreiben eines PASCAL-Textes) sowie Hinweise auf maschinenabhängige Implementationen (für mehr als 100 verschiedene Rechenanlagen kleinerer und größter Computer-Hersteller). Anfang 1979 hatte die Vereinigung weltweit ca. 3000 Mitglieder. Da das Hauptziel erreicht ist, wird z. Zt. intern über eine neue Form des Weiterbestehens diskutiert.

Der schon oben erwähnte PASCAL-P-Compiler – ebenfalls in Zürich entstanden (1973) – darf als ein Meilenstein in der Geschichte von PASCAL betrachtet werden. Er diente nicht nur der Verbreitung dieser Programmiersprache, sondern gab auch Anstöße zur Realisation neuer Rechner-Strukturen. Mit der PASCAL-Microengine von Western Digital ist zum erstenmal ein Computer auf dem Markt, dessen Prozessor den (modifizierten) P-Code direkt decodiert und ausführt. Dieser Rechner braucht und hat keinen (!) Assembler. Die gesamte Software für das Betriebssystem (Text-Editor usw.) ist in PASCAL geschrieben und liegt als P-Code auf der Magnetplatte. Man darf gespannt sein, wie andere Hersteller von Rechnern auf diese Herausforderung reagieren.

Das in dieser Arbeit beschriebene PASCAL ermöglicht wie die Sprachen FORTRAN, COBOL, ALGOL oder BASIC das Programmieren sequentieller Prozesse. Die besondere Bedeutung von *parallelen* Prozessen, wie sie bei der Echtzeitverarbeitung (Multiprogrammierung, Betriebssysteme, Synchronisation verschiedener Prozesse u. a.) auftreten, führte zur Entwicklung von „CONCURRENT PASCAL" [10]. Eine ebenfalls auf dem sogenannten Monitor-Konzept – ein Monitor koordiniert zwei oder mehrere Prozesse – aufgebaute Sprache wurde 1977 von *N. Wirth* vorgestellt; ihre Nachfolgerin MODULA-2 [11] befindet sich noch im experimentellen Stadium.

Die Entwicklung der Sprache – im wesentlichen – durch eine einzelne Person ist ein weiteres Merkmal von PASCAL. Dadurch hat PASCAL ein hohes Maß an innerem Zusammenhang. Auf der anderen Seite wurde dadurch manchen Gesichtspunkten und Wünschen, wie beispielsweise einem wahlfreien Zugriff auf Dateien, nicht Rechnung getragen. Die internationale Standardisierung wird hier sicher die Gewichte zwischen den Anforderungen der Lehre und der industriellen Praxis neu überdenken.

2 Überblick über PASCAL

Ein Programm dient im weitesten Sinne zur Manipulation von Daten und beinhaltet deshalb zwei Aspekte: zum einen die Form der Daten und zum anderen die Art, wie diese Daten verarbeitet werden sollen. Für das erstere bietet PASCAL ein reichhaltiges Angebot, dem Problem angemessene Datenstrukturen zu konstruieren (*Bild 1*). Das reicht von einfachen Datentypen mit den sogenannten Standardtypen (*integer, real, boolean* und *char*) zu den zusammengesetzten für Felder (*array*), Mengen (*set*), Dateien (*file*) und Verbunde (*record*). Über viele andere Programmiersprachen hinausgehend gibt es in PASCAL die Aufzählungstypen und die Unterbereichstypen. Mit den Aufzählungstypen ist es möglich, die meist undurchsichtige Verschlüsselung von Daten zu verlassen; dies übernimmt bei PASCAL der Compiler. Statt einer Zuordnung für die Wochentage, z. B. Montag sei 0, Dienstag sei 1 usw., bis Sonntag sei 6, vergibt man Namen und ordnet sie in der gewünschten Reihenfolge, also z. B.: MONTAG, DIENSTAG usw. Die sogenannten Unterbereichstypen dienen besonders der Zuverlässigkeit eines Programms, wenn der Wertebereich auf die sinnvoll vorkommenden Werte beschränkt wird, Beispiel: die Anzahl der Tage eines Monats bewegt sich nur zwischen 1 und 31.

Während Felder und Dateien wohlbekannte Typen sind, erlaubt das Arbeiten mit Mengen elegante Lösungen (ein Buchstabe ist z. B. in der Menge der Vokale enthalten). Ein Verbund besteht im Gegensatz zu anderen zusammengesetzten Datentypen aus Komponenten verschiedenen Typs. Ein übliches Beispiel ist die Personenbeschreibung für eine Kartei. Ein einzelnes Karteiblatt kann in PASCAL z. B. durch einen Verbund aus Personenname mit Vor- und Zuname, Alter, Geschlecht, Wohnort und Postleitzahl, Familienstand und anderem dargestellt werden. Um Speicherplatz – auf Kosten der Verarbeitungsgeschwindigkeit – zu sparen, besteht die Möglichkeit,

Tabelle 1. Aufstellung aller Grundsymbole der Sprache PASCAL

1. Buchstaben
****

A	B	C	D	E	F	G	H	I	J
K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z				
a	b	c	d	e	f	g	h	i	j
k	l	m	n	o	p	q	r	s	t
u	v	w	x	y	z				

2. Ziffern
****

0	1	2	3	4	5	6	7	8	9

3. Sonderzeichen
****

+	-	*	/	>	<	>=	<=	=	<>
.	,	;	:	'	(	)	[	]	{	}
␣	:=	↑							

4. Wortsymbole (reservierte Namen)
****

AND	ARRAY	BEGIN	CASE	CONST	DIV
DO	DOWNTO	ELSE	END	FILE	FOR
FUNCTION	GOTO	IF	IN	LABEL	MOD
NIL	NOT	OF	OR	PACKED	PROCEDURE
PROGRAM	RECORD	REPEAT	SET	THEN	
TO	TYPE	UNTIL	VAR	WHILE	WITH

Anhang: Ersatzdarstellungen
****

Grundsymbol	Ersatzsymbol
↑	^ oder @
>	GT
<	LT
>=	GE
<=	LE
<>	NE
:	%
:	%= oder .=
;	,'
[ ]	(. .)
{ }	(* *)

Programm

```
PROGRAM  HISTOGRAMM(INPUT,  OUTPUT);
CONST
                OFFSET  =  2048;
            SKALIERUNG  =  0.02672  { milli-VOLT };
TYPE
                 IFELD  =  ARRAY  [0..50]  OF  INTEGER;
VAR
     HAEUFIGKEITSVEKTOR:  IFELD;
            KENNZEICHEN:  ALFA{ =PACKED ARRAY[1..10] OF CHAR};
                      N,
          KLASSENANZAHL,
                    MUE:  INTEGER;
           PCMDATENWORT:  0..4095  {  = 2**12 - 1  };
             ANFANGSLAGE,
          KLASSENBREITE,
               MESSWERT,
                  SUMME:  REAL;
PROCEDURE  KLASSIFIZIERE(          Y,
                              ANFANG,
                         KLSBREITE:  REAL;
                         KLSANZAHL:  INTEGER;
                               VAR  H:  IFELD);
   VAR
                      LAGE:  INTEGER;
   BEGIN  {KLASSIFIZIERE}
      LAGE  :=  TRUNC((Y  -  ANFANG  +  KLSBREITE)
                    /  KLSBREITE);
      IF  LAGE  <=  0
      THEN
        H[0]  :=  H[0]  +  1  { "Y" IN UNTERER RESTKLASSE }
      ELSE
         IF  LAGE  <=  KLSANZAHL
         THEN
           H[LAGE]  :=  H[LAGE]  +  1
         ELSE  { "Y" IN OBERER RESTKLASSE }
           H[KLSANZAHL  +  1]  :=  H[KLSANZAHL  +  1]  +  1
    END  {KLASSIFIZIERE};
PROCEDURE  DRUCKEHISTOGRAMM(VAR  AUS:  TEXT;
                            FAKTORH,
                         KLSANZAHL:  INTEGER;
                                 H:  IFELD;
                            ANFANG,
                         KLSBREITE:  REAL);
   EXTERN;
BEGIN  {HISTOGRAMM}
   KLASSENANZAHL  :=  45;
   ANFANGSLAGE  :=  -  1.5;
   KLASSENBREITE  :=  0.2;
   FOR  MUE  :=  0  TO  KLASSENANZAHL  +  1  DO
      HAEUFIGKEITSVEKTOR[MUE]  :=  0;
   SUMME  :=  0.0;
   N  :=  0;
   FOR  MUE  :=  1  TO  10  DO
      READ(INPUT,  KENNZEICHEN[MUE]);
   READLN(INPUT);
   WHILE  NOT  EOF(INPUT)  DO
      BEGIN
         N  :=  N  +  1;
         READ(INPUT,  PCMDATENWORT);
         MESSWERT  :=  (PCMDATENWORT  -  OFFSET)  *  SKALIERUNG;
         SUMME  :=  SUMME  +  MESSWERT;
         KLASSIFIZIERE(MESSWERT,  ANFANGSLAGE,  KLASSENBREITE,
            KLASSENANZAHL,  HAEUFIGKEITSVEKTOR)
      END  {WHILE};
   PAGE(OUTPUT);
   WRITELN(OUTPUT,  '      VERSUCHS-KENNZEICHEN=',  KENNZEICHEN);
   WRITELN(OUTPUT,  '      ANZAHL DER MESSWERTE=',  N);
   WRITELN(OUTPUT,  '      MITTELWERT=            ',  SUMME  /  N:
      10:  2);
    DRUCKEHISTOGRAMM(OUTPUT,  24,  KLASSENANZAHL,
      HAEUFIGKEITSVEKTOR,  ANFANGSLAGE,  KLASSENBREITE)
END  {HISTOGRAMM}.
```

Eingabe (input)

```
C4117BF
2100 2103 2104 2108 2112 2114 2115 2118 2121 2124 2126 2125 2126 2127
2128 2134 2138 2140 2136 2133 2127 2122 2118 2112 2108 2102 2096 2090
2083 2081 2084 2087 2086 2086 2088 2091 2090 2092 2095 2094 2091 2088
2083 2080 2079 2079 2080 2076 2071 2068 2068 2067 2069 2071 2071 2071
2068 2064 2060 2057 2060 2067 2072 2077 2081 2082 2080 2080 2086 2096
2104 2108 2112 2116 2123 2126 2126 2127 2132 2138 2144 2153 2162 2171

2173 2167 2162 2162 2164 2166 2165 2164 2168 2175 2182 2184 2188 2195
2199 2198 2195 2192 2184 2176 2170 2168 2172 2175 2175 2174 2172 2168
2164 2160 2160 2164 2170 2176 2176 2174 2175 2180 2187 2192 2192 2187
2182 2180 2182 2181 2179 2181 2187 2190 2189 2188 2191 2192 2190 2185
2180 2175 2167 2160 2162 2168 2172 2172 2168 2169 2175 2182 2185 2187
```

Ausgabe (output)

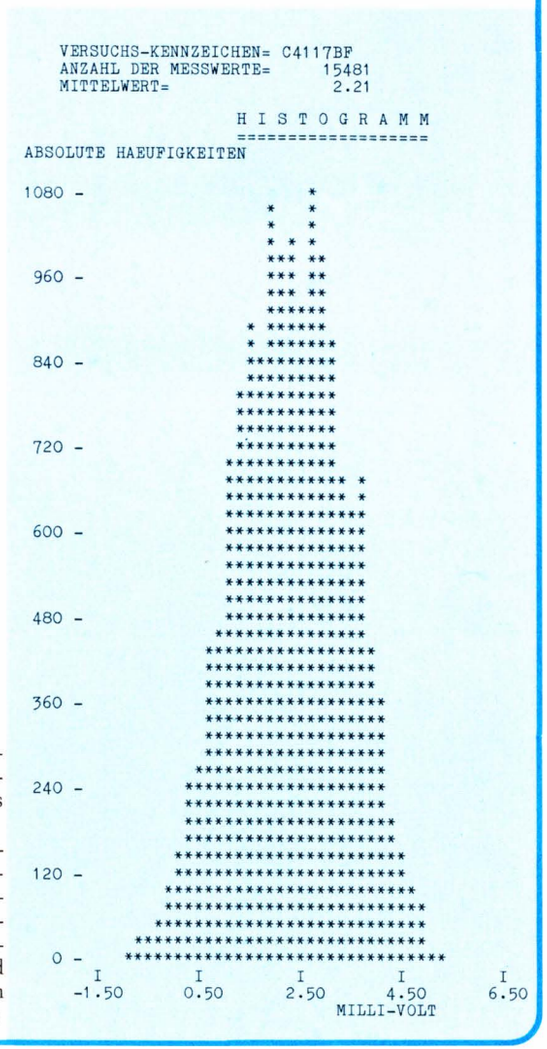


Bild 2. Programmbeispiel HISTOGRAMM, das aus PCM-Meßwerten (*input*) eines physiologischen Experimentes den Mittelwert und die relativen Häufigkeiten berechnet und das Histogramm ausdruckt (*output*)

die strukturierten Datentypen zu packen, was nicht nur auf Kleinrechnern nützlich sein kann. Die Zeichenkette (*string*) nimmt unter den gepackten Feldern eine besondere Stellung ein; über die Beschränkung auf eine feste Länge wird im Abschnitt 7.2 über Textverarbeitung noch zu reden sein. Für die Bearbeitung dynamischer Informationsstrukturen (Graphen) steht der Zeigertyp zur Verfügung. Die Zeiger sind an einen Datentyp fest gebunden und verketten durch ihre Verweise die Objekte dieser Typen zu linearen oder baumartigen Graphen.

Zur Steuerung des Programmablaufs gibt es in PASCAL alle notwendigen Mechanismen. Neben den einfachen (die Wertzuweisung erfolgt wie in ALGOL-60 mit dem Operator „:=") gibt es die sogenannten strukturierten Anweisungen. Dazu gehören die einseitige und zweiseitige Verzweigung (*if... then... else*), die Zählschleife (*for... to... do*), die Bedingungsschleifen (*repeat...until* und *while...do*), die Fallunterscheidung (*case...of...end*) und auch den unbedingten Sprung (*goto*). Zusammengehörende Anweisungen – beispielsweise in einer Zählschleife – werden mit *begin* und *end* eingeklammert. Für den Zugriff auf Komponenten der oben erwähnten Verbunde existiert eine Inspektionsanweisung (*with... do*).

PASCAL hat eine ausgefeilte Unterprogrammtechnik in Form von Prozeduren und Funktionen; als Prozedur- (bzw. Funktions-)Parameter können Werte und Variable sowie Prozeduren und Funktionen selbst auftreten. Die meisten Compiler gestatten es, externe Unterprogramme – auch anderer Sprachen – in das Programm einzuschleusen (Anmerkung: da ALGOL-60 auch die sogenannten Namen-Parameter kennt, stößt die Verwendung von ALGOL-Bibliotheken auf Schwierigkeiten). Für die Standardtypen gibt es eine Reihe von Standardfunktionen (z. B. Exponential-, Nachfolgerfunktion u. a.) und Standardprozeduren (z. B. für die Ein-/Ausgabe: *read*, *write* u. a.).

Jede Prozedur und Funktion hat wie das Programm selbst einen Vereinbarungs- und Anweisungsteil; beide Teile bilden einen Block. Die Strenge der Sprache bringt es mit sich, daß jede Variable vereinbart und von einem bestimmten Typ – gegebenenfalls durch den Programmierer selbst definiert – sein muß. Davon sind natürlich die vordefinierten Namen (Bezeichner) ausgenommen.

Ein konkretes Programmbeispiel möge dem Leser einen ersten Eindruck über ein mittleres PASCAL-Programm vermitteln (*Bild 2*). Das Programm „HISTOGRAMM" ermittelt ein Histogramm von Versuchsdaten eines physiologischen Experimentes. Die Meßwerte wurden mit einem PCM-Magnetbandgerät aufgezeichnet und über die Digitalkassette als Zwischenträger einer Großrechenanlage eingegeben. Das Programm liest die Daten ein, die sich wegen dem 12-Bit-A/D-Umsetzer nur zwischen 0 und 4095 bewegen, klassifiziert sie und gibt die relativen Häufigkeiten grafisch auf den Drucker aus. Das Unterprogramm zur Ausgabe wurde in eine Bibliothek ausgelagert, um a) diese Möglichkeit zu demonstrieren und b) hier Platz zu sparen. In „HISTOGRAMM" sind nur einige spezifische Eigenschaften von PASCAL enthalten. Das äußere Erscheinungsbild wurde mit einem Schönschreibprogramm aus der Bibliothek hervorgerufen.

Wegen der präzisen – natürlich mit kleinen Einschränkungen – Ausdrucksweise für Algorithmen und den dazu gehörenden Datenstrukturen findet man in der Literatur schon häufig Problemlösungen in PASCAL dargestellt, auch dann, wenn die eigentliche Verarbeitung mit einer anderen Sprache erfolgt. Solange es für die graphische Darstellung von Datentypen keine den Programmablaufplänen vergleichbare Form gibt, werden „lesbare" Beschreibungen den Vorzug haben.

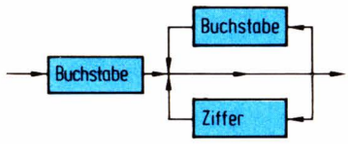
3 Die Elemente der Programmiersprache PASCAL

3.1 Grundsymbole, Bezeichner

PASCAL kennt – wie andere Sprachen – einen festen Satz von Zeichen. Per Definition ist ein Grundsymbol entweder ein Buchstabe, eine Ziffer, oder ein Sonderzeichen. In der *Tabelle 1* sind alle Grundsymbole verzeichnet. Daß die hier gezeigten Programmbeispiele keine Kleinbuchstaben enthalten, möge der Leser nicht als Vorentscheidung für eine Norm auslegen (der Verfasser hält die Möglichkeit der Groß- und Kleinschreibung in einem PASCAL-Programm für zweckmäßig; die mengenmäßige Ausstattung der Rechenanlagen mit Zeichen sollte sich am menschlichen Maß orientieren).

Diejenigen Grundsymbole der Sprache PASCAL, die aus mehreren Buchstaben bestehen, nennt man Wortsymbole. Sie sind reservierte Namen mit einer festen Bedeutung, die nicht verändert werden kann. In der Literatur und in den Lehrbüchern sind sie meist fettgedruckt, um sie von anderen Bezeichnern zu unterscheiden. Es ist also z. B. verboten, den Namen RECORD einer selbstdefiniertên Größe zu geben. Aus technischen Gründen gibt es für einige Grundsymbole Ersatzdarstellungen, z. B. sind der Doppelpunkt und das Prozentzeichen auf vielen Anlagen im Programmtext gleichwertige Zeichen. Einige Operatoren dürften dem Leser vertraut sein. Es wird vielleicht das Zeichen für die Exponentation vermißt, was schon heftige Diskussionen ausgelöst hat. Der Grund für ihr Weglassen liegt vor allem daran, daß es eine teure Funktion wäre. In einem später folgenden Programm (*Bild 26*) wird eine Möglichkeit ihrer Realisation gezeigt. Dem Fehlen dieser und auch anderer Eigenschaften in PASCAL liegt der Grundsatz zugrunde, daß sich der Programmierer über die Dinge, die er tut, ausdrücklich im klaren ist. (Bei der Berechnung von $(-3)^{5/2}$ kämen beispielsweise plötzlich die komplexen Zahlen ins Spiel, falls nicht nur positive Basiswerte potenziert werden.) Ein PASCAL-Programm verwendet daher auch möglichst wenig Voreinstellungen. Das kostet in der Entwicklungsphase ein Mehr an Zeit, dafür ist die Wartung erheblich erleichtert.

Alle Größen, mit denen in einem Programm „gerechnet" wird, erhalten einen Namen; das sind Variablen, Konstanten, Typen, Prozeduren und Funktionen. Diese Bezeichner müssen bestimmten Regeln ge-



◀ Bild 3. Syntax-Diagramm für einen PASCAL-Bezeichner

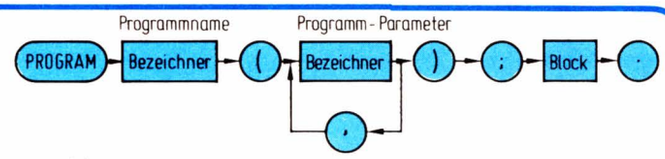


Bild 4. Diagramm für die Syntax eines PASCAL-Programms

horchen. Für die Darstellung dieser Syntax-Regeln haben sich die sogenannten Syntax-Diagramme bestens bewährt. Man findet sie fast in allen Büchern über PASCAL [5, 7, 8]. Wegen seiner exakten Darstellung ist [5] an dieser Stelle hervorzuheben, weil es für den fortgeschrittenen Programmierer die Sprachen ALGOL, FORTRAN und PASCAL einheitlich darstellt. *Bild 3* zeigt das erste dieser Diagramme für einen PASCAL-Bezeichner. Man liest sie entsprechend der Pfeilrichtung und prüft an den Verzweigungspunkten, ob die untersuchte Einheit (für Bild 3 ein Bezeichner) den gezeichneten Pfaden und Kästchen folgt. Ein Bezeichner beginnt mit einem Buchstaben, dem dann abwechselnd Buchstaben und Ziffern folgen können.

So sind zulässige Namen:
N
GAMMA
FAKULTAET
KLASSIFIZIERE
M1
M1B
ZZZ999
KLASSIFIKATION

Falsch wären folgende Bezeichner:
K.HUBER
OTTO MAYER
3.

Für die Länge eines Namens gilt die Empfehlung, daß mindestens die ersten acht Zeichen vom Compiler als relevant angesehen werden. Die Wahl eines Namens soll sich an der leichten Lesbarkeit des Programmes und nicht an einer schnellen „Schreibbarkeit" orientieren. Da die meisten Compiler nur eine beschränkte Anzahl der ersten Zeichen eines Bezeichners auswerten (und den Rest wegwerfen), muß man die Namen so wählen, daß keine Verwechslungen auftreten. Für einen (Schmalspur-)Compiler, der nur acht Zeichen eines Namens beachtet, wären KLASSIFIERE und KLASSIFIKATION ein Bezeichner KLASSIFI.

3.2 Der Aufbau eines PASCAL-Programms

Ein PASCAL-Programm besteht grundsätzlich aus zwei Teilen: einem Programmkopf und einem Block. Das entsprechende Syntax-Diagramm zeigt *Bild 4*. Nach dem Wortsymbol „*program*" folgt ein Programmname; zwei Namen müssen durch ein Leerzeichen getrennt sein – es dürfen natürlich auch mehrere sein oder ein bzw. mehrere Zeichenwechsel (für das Leerzeichen steht in Tabelle 1 das Symbol „␣").

Ein Block ist, wie *Bild 5* zeigt, ein umfangreiches Gebilde. Nacheinander folgen die Vereinbarungsteile für Marken, Konstanten, Typen, Variablen, Prozeduren, Funktionen und schließlich der Anweisungsteil mit *begin* und *end* geklammert. Wenn man die Pfade verfolgt, sieht man, daß es auch Blöcke ohne jegliche Vereinbarungen geben kann. Nehmen wir schließlich noch die sogenannte leere Anweisung vorweg, so können wir bereits ein einfaches und lauffähiges Programm schreiben (*Bild 6*). Als Programm-Parameter nach dem Namen PRIMITIV dient OUTPUT. *output* – ein Standardname – bezeichnet eine sogenannte Text-Datei, die neben dem Zeichensatz der Anlage die Symbole für den Zeilen- und Seitenwechsel enthält. In dieser Datei erfolgt normalerweise die Programmausgabe, daher ist die Angabe von *output* in der Regel vorgeschrieben. Entsprechend dazu gibt es für die Eingabe die Standarddatei *input*. Beide Parameter reichen vorerst für die Ein- und Ausgabe der Daten. Mit der Aufzählung von OUTPUT ist beim Programmlauf die Eröffnung einer Datei mit gleichem Namen verbunden. Trotz der Kürze von PRIMITIV geschieht also etwas, da aber kein Zeichen ausgegeben wird, bleibt die Datei *output* leer.

Nach Einführung der Standardprozedur „write" kann man zur Demonstration ein Programm schreiben, das einen arithmetischen Ausdruck zweier Zahlen berechnet und das Resultat ausdruckt. Was mit WRITE ausgegeben werden soll, steht als Parameter zwischen den runden Klammern. Es kann eine Konstante, ein Ausdruck, der Wert einer Variablen, ein Funktionswert und auch direkt eine Zahl (Zeichen, logischer Wert, Zeichenkette) sein. *Bild 7* zeigt das Programm mit dem ausgedruckten Ergebnis. Für die Berechnung selbst braucht man natürlich keinen Computer zu bemühen.

Von diesen einfachen Programmen abgesehen sind für Berechnungen veränderliche Größen notwendig.

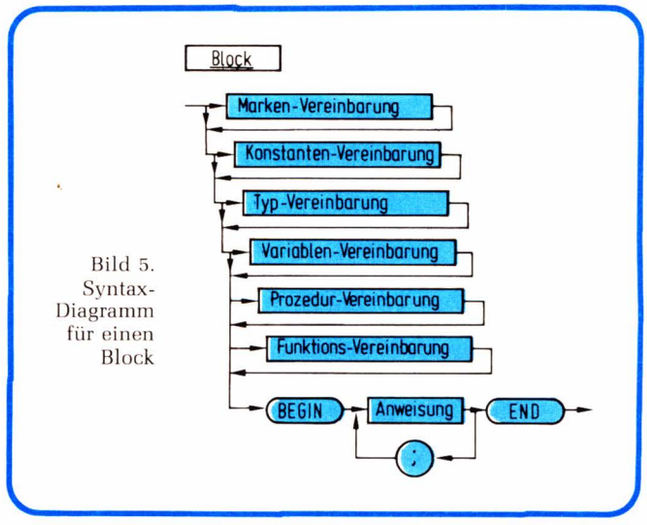


Bild 5. Syntax-Diagramm für einen Block

Programm

```
PROGRAM PRIMITIV(OUTPUT);
BEGIN
END.
```

Ausgabe *(output)*

Bild 6. Beispiel für ein minimales PASCAL-Programm; es enthält keine Vereinbarungen, aber eine (sogenannte) leere Anweisung

Programm

```
PROGRAM  UNMITTELBAR(OUTPUT);

BEGIN
   WRITE(533  +  186)
END  {UNMITTELBAR}.
```

Ausgabe *(output)*

```
719
```

Bild 7. Ein einfaches Programm ohne Vereinbarungsteil, das einen arithmetischen Ausdruck mit ganzen Zahlen berechnet und ausdruckt. Da keine Zahlen eingegeben werden, ergibt sich bei jedem Programmlauf das gleiche Ergebnis. Entsprechendes wäre auch für reellwertige und logische Ausdrücke möglich

Als erstes soll daher von dem Vereinbarungsteil die Variablen-Vereinbarung betrachtet werden (*Bild 8*). Hier erhalten nach dem Wortsymbol „var" alle Variablen einen Namen, dem nach einem Doppelpunkt eine Typenangabe erfolgt. Die volle Darstellung für einen Typ wird in *Bild 16* (s. u.) beschrieben. Zunächst sollen nur die einfachen Datentypen benutzt werden.

3.3 Einfache Datentypen

Vordefiniert sind als Bezeichner für die Standardtypen:

integer	für ganze Zahlen
real	für Gleitkommazahlen
boolean	für logische Variable
char	für Zeichen

Die Zahlenbereiche für „*integer*" und „*real*" hängen stark von der Bitlänge des Rechners ab; durch spezielle Routinen lassen sich bei Bedarf höhere Genauigkeiten erzielen. Die Anzahl der Zeichen des Datentyps „*char*" wird von dem Zeichenvorrat der Anlage bestimmt, der natürlich über den Zeichensatz von PASCAL (Tabelle 1) hinausgehen kann. Der Typ „*boolean*" besitzt zwei logische Werte: „*false*" und „*true*". Beides sind Standardnamen (Tabelle 2, s. u.) und keine Wortsymbole. Es wäre also möglich, diese Namen – wie schon „boolean", „integer", „real" und „char" – in einem völlig anderen Zusammenhang zu verwenden! Auch die ganzen und reellen Zahlen (exakter: die Teilmengen davon) gehorchen bestimmten Regeln; siehe dazu die Syntax-Diagramme in Bild *9a* und *9b*. Beispiele für reelle Zahlen sind

0.02672 +13.81 –509 23E-7
(E bezeichnet den Zehnerexponenten).

Die Werte des Datentyps *char* werden mit Anführungszeichen eingeklammert:

'A' '.' '1'

Mehrere Zeichen aufeinanderfolgend bilden eine Zeichenkette *(string)*:

'PASCAL'
'DIES IST AUCH EINE ZEICHENKETTE'
''''
'K. HUBER'

Eine Zeichenkette ist also von einem Namen völlig verschieden. Sie besitzt wie eine Zahl einen Wert, der manipuliert werden kann.

Doch nochmal zurück zu der Variablen-Vereinbarung. Mit den nun bekannten Typen kann man vollständige Vereinbarungen hinschreiben:

VAR ANTWORT:BOOLEAN;
oder
VAR ZEIT:REAL; u. a

Wie an den nachfolgenden Programmbeispielen zu sehen ist, schaffen größere Abstände zwischen den Namen und den Sonderzeichen mehr Übersichtlichkeit. Die Konstanten-Vereinbarung dient zur Definition konstanter Größen (*Bild 10*); wie z. B. für die Zahl π, die Lichtgeschwindigkeit c. Als „Konstante" kommen ganze, reelle Zahlen, Zeichen, Zeichenketten und die logischen Werte in Betracht. Beispiele:

```
CONST PI     = 3.1415926; JMAX = 100;
      C      = 2.9979E+8 ( * in m/s * );
      FRAGE  = 'WOLLEN SIE WEITERMACHEN?'
```

Die rechte Seite der Definition – sie ähnelt der Wertzuweisung –, bestimmt den Datentyp. Im Unterschied zu den Variablen kann der Wert einer Konstanten im

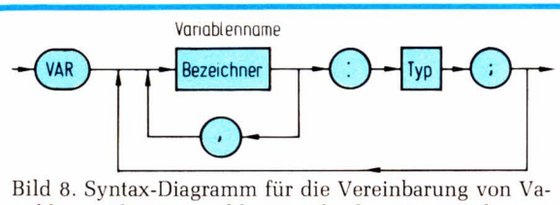


Bild 8. Syntax-Diagramm für die Vereinbarung von Variablen. Mehrere Variable vom gleichen Typ werden mit Kommas getrennt. Zu beachten ist, daß vor dem Typbezeichner (z. B. *integer*) ein Doppelpunkt steht

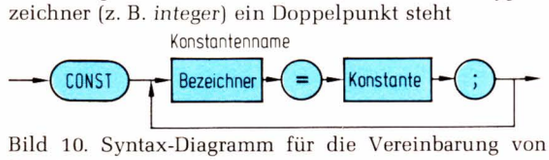


Bild 10. Syntax-Diagramm für die Vereinbarung von Konstanten

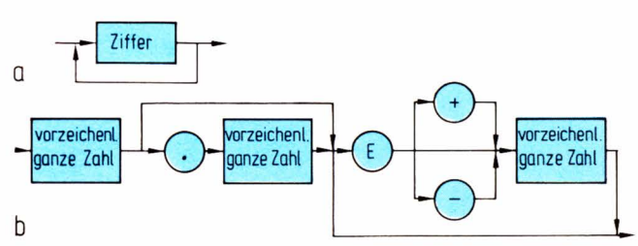


Bild 9. Die Syntax-Diagramme für eine vorzeichenlose ganze Zahl (9a) und für eine vorzeichenlose Zahl (9b). Durch ein vorausgehendes Vorzeichen erhält man die ganzen bzw. reellen Zahlen (*integer* bzw. *real*); kein Vorzeichen bedeutet wie gewohnt ein Pluszeichen

Programm

```
PROGRAM  VERZOEGERUNGSGLIED(OUTPUT);
   { Reaktion auf einen Einheitssprung;
     Version A ohne externe Eingabe }

CONST
        ZEITKONSTANTE  =  5.0  { sec };

VAR
                    ZEIT,
                AUSGANG:  REAL;

BEGIN
   ZEIT  :=  15.0;
   AUSGANG  :=  1.0  -  EXP(-  ZEIT  /  ZEITKONSTANTE);
   WRITE(AUSGANG)
END  {VERZOEGERUNGSGLIED}.
```

Bild 11. Programm zur Berechnung der Übergangsfunktion eines Verzögerungsgliedes 1. Ordnung

Ausgabe *(output)*

```
9.5021293163214E-001
```

Bild 11a. Berechnung von $1\text{-}e^{-t/T}$ für t=15 und Ausgabe des reellen Ergebnisses

Programm

```
PROGRAM  VERZOEGERUNGSGLIED(OUTPUT);
   { Reaktion auf einen Einheitssprung;
     Version B ohne externe Eingabe,
     aber mit formattierter Ausgabe }

CONST
        ZEITKONSTANTE  =  5.0  { sec };

VAR
                    ZEIT,
                AUSGANG:  REAL;

BEGIN
   ZEIT  :=  15.0;
   AUSGANG  :=  1.0  -  EXP(-  ZEIT  /  ZEITKONSTANTE);
   WRITE(AUSGANG:  12)
END  {VERZOEGERUNGSGLIED}.
```

Ausgabe *(output)*

```
9.502E-001
```

Bild 11b. Durch „AUSGANG: 12" in der WRITE-Anweisung ist die Länge der Ausgabe auf 12 Zeichen formatiert

Eingabe *(input)*

```
25.0
```

Programm

```
PROGRAM  VERZOEGERUNGSGLIED(INPUT,  OUTPUT);
   { Reaktion auf einen Einheitssprung;
     Version C mit externer Eingabe }

CONST
        ZEITKONSTANTE  =  5.0  { sec };

VAR
                    ZEIT,
                AUSGANG:  REAL;

BEGIN
   READ(ZEIT);
   AUSGANG  :=  1.0  -  EXP(-  ZEIT  /  ZEITKONSTANTE);
   WRITELN('      ZEIT=            ',  ZEIT:  12:  4);
   WRITELN('      AUSGANGSGROESSE=',  AUSGANG:  12:  4)
END  {VERZOEGERUNGSGLIED}.
```

Ausgabe *(output)*

```
ZEIT=                  25.0000
AUSGANGSGROESSE=        0.9933
```

Bild 11c. Hier wird nun der Wert für t durch READ(ZEIT) eingelesen; es erfolgt keine Prüfung, ob $t \geqq 0$ ist (nur dafür gilt die Berechnung!). Die Ausgabe wurde zusätzlich verbessert: Zur Erläuterung werden Zeichenketten mit ausgedruckt und das Format der reellen Zahlen von der Gleit- auf die Festpunktdarstellung umgeschaltet. *writeln* gibt wie *write* die Parameter in den runden Klammern aus, und bewirkt zusätzlich einen Zeilenwechsel

gesamten Programm nicht verändert werden. Bis auf diese gewollte Eigenschaft kann mit einer Konstanten wie mit einer Variablen gleichen Typs verfahren werden. Die in der dritten Vereinbarung (C = 2.99...) auftretenden Zeichen („(★", „★)") sind die sogenannten Kommentarklammern, die im Programmtext ohne Wirkung für den Berechnungsablauf eingeschoben werden können. Sie dürfen aber nicht in Namen und Zahlen stehen.

Das nächste Programmbeispiel (*Bild 11a*) enthält schon beide Vereinbarungsmöglichkeiten zur Berechnung der Übergangsfunktion. In der Version A wird der Variablen ZEIT mit dem Zeichen „:=" ein Wert zugewiesen. In der nächsten Zeile erfolgt die Berechnung mit der Exponentialfunktion (standardmäßig vorhanden), wobei vor der Auswertung der Funktion der Ausdruck in den runden Klammern berechnet wird. AUSGANG erhält schließlich eine reelle Zahl zugewiesen, die mit der Standardprozedur *write* ausgegeben wird. Ohne weitere Spezifikation wird eine „real"-Größe in der vollen Genauigkeit ausgedruckt. Braucht man nicht so viele Stellen, so kann man durch das Anhängen einer ganzen Zahl (oder einer ganzzahligen Variable) die Länge der Zahldarstellung im Gleitpunktformat festlegen. Im Bild 10b bedeutet die Angabe AUSGANG : 12, daß insgesamt zwölf Zeichen gedruckt werden. In Bild 11c wurde nun das Programm dahingehend abgeändert, daß Zahlenwerte zur Berechnung eingegeben werden können. Dazu hilft die Standardprozedur *read*, die hier wegen der reellen Variable ZEIT eine reelle Zahl erwartet und den eingelesenen Wert dann ZEIT zuweist. Um die Eingabe zu kontrollieren, wird dieser Wert wieder ausgegeben. Statt *write* findet sich nun *writeln* im Programmtext, was zusätzlich am Ende der Ausgabe einen Zeilenwechsel bewirkt. In den runden Klammern befindet sich als erstes eine Zeichenkette, die auch zuerst ausgegeben wird. Nach einem Komma, das also mehrere Ausgabe-Parameter trennt, folgt die Variable ZEIT mit einer weiteren Formatspezifikation: ZEIT: 12: 4. Dies ergibt eine Festpunktzahl mit insgesamt zwölf Zeichen einschließlich dem Vorzeichen, vier Zeichen sind davon rechts vom Komma (Punkt in angloamerikanischer Schreibweise). Diese Programme enthalten schon mehrere Anweisungen, die durch das Semikolon – nicht durch den Zeilenwechsel – getrennt sind. Vor *end* oder *repeat* braucht kein Semikolon zu stehen.

Den Umgang mit den Variablen vom Typ *char* und *boolean* möge ein Programmbeispiel („BUCHSTABENRATEN" in *Bild 12*) aufzeigen: Mit einem selbst vorgegebenen Buchstaben soll der gesuchte Buchstabe erraten werden. Zur Vereinfachung wird der gesuchte Buchstabe ins Programm gesetzt – in einer modifizierten Version könnte ein Zufallsgenerator einen Buchstaben auswählen. Die Variable mit dem Namen GESUCHT ist im Vereinbarungsteil wie GEGEBEN (s. u.) von dem einfachen Standard-Datentyp *char*. Sie erhält zu Beginn des Anweisungsteils den festen Wert 'H' zugewiesen. Die (einfachen) Anführungszeichen umklammern den Buchstaben H und unterscheiden dadurch 'H' von einer – hier nicht – definierten Variablen H. Durch den Einlesevorgang bekommt GEGE-

BEN ebenso einen Wert eines Buchstabens (genau genommen eines Zeichens aus dem gesamten Zeichenvorrat; die Richtigkeit der Eingabe wird zunächst nicht überprüft). Da nun der Zeichenvorrat geordnet und insbesondere die Buchstaben alphabetisch gereiht sind, können Zeichen-Variable miteinander verglichen werden:

GEGEBEN < GESUCHT

ist somit ein Vergleich, ob sich der eingegebene Buchstabe im Alphabet vor dem gesuchten befindet. Das Resultat ist ein logischer Wert (FALSE oder TRUE), der in unserem Beispiel der Variablen ANTWORT zugewiesen wird. An Hand der Ausgabe kann durch wiederholtes Probieren der gesuchte Buchstabe – natürlich nicht vom Programmierer selbst – erraten werden. Durch das Anwenden weiterer PASCAL-Eigenschaften wird man später das Programm verbessern können. (Anmerkung: Da nicht alle Codes die Buchstaben exakt alphabetisch ordnen (z. B. enthält der EBCDI-Code eine Lücke zwischen I und J sowie zwischen R und S), muß man auf den maschinenabhängigen Zeichenvorrat für den Typ *char* achten).

3.4 *Aufzählungstypen und Unterbereichstypen*

Sie gehören nach der im Bild 1 (Abschnitt 1) vorgenommenen Einteilung zu den einfachen Datentypen. Wegen ihrer besonderen Bedeutung wird ihnen ein eigener Abschnitt gewidmet. Beide Formen dienen dazu, ein Programm

- lesbarer,
- sicherer gegenüber falschen Eingaben und
- verständlicher in bezug auf den verwendeten Algorithmus zu machen.

Zunächst zu den Aufzählungstypen (auch Skalartypen genannt): ein solcher Typ wird vom Programmierer selbst definiert. In runden Klammern werden der Reihe nach die Werte mit Namen (Bezeichnern) aufgelistet. Z. B.:

Eingabe *(input)*

```
C
```

Programm

```
PROGRAM  BUCHSTABENRATEN(INPUT,  OUTPUT);

VAR
                GESUCHT,
                GEGEBEN:  CHAR;
                ANTWORT:  BOOLEAN;

BEGIN
   GESUCHT  :=  'H';
   READ(GEGEBEN);
   ANTWORT  :=  GEGEBEN  <  GESUCHT;
   WRITE('  EINGEGEBENES  "',  GEGEBEN);
   WRITLEN('"  VOR DEM GESUCHTEN BUCHSTABEN ?',  ANTWORT)
END  {BUCHSTABENRATEN}.
```

Ausgabe *(output)*

```
EINGEGEBENES  "C"  VOR DEM GESUCHTEN BUCHSTABEN ?      TRUE
```

Bild 12. Programm BUCHSTABENRATEN. Das Raten ist natürlich nur dann spannend, wenn der Programmtext verdeckt ist. Durch wiederholtes Probieren kreist man den gesuchten Buchstaben ein

(STOP, AUFNAHME, WIEDERGABE, LOESCHEN, VORLAUF, RUECKLAUF)
(POSITIV, NEGATIV)
(ROT, ORANGE, GELB, GRUEN, BLAU, VIOLETT)
(MON, DIE, MIT, DON, FRE, SAM, SON)
(KOENIG, DAME, TURM, LAEUFER, SPRINGER, BAUER)

Da wir bisher nur die Variablen-Vereinbarung kennen, könnten wir eine Variable FLANKE wie folgt definieren:

VAR FLANKE: (POSITIV, NEGATIV);

An dieser Stelle ist es zweckmäßig, kurz innezuhalten und die noch öfters erscheinenden Typ-Vereinbarung einzuführen (*Bild 13*). Dazu ist das Syntax-Diagramm für den einfachen Datentyp (*Bild 14*) nachzutragen, das mit *Bild 16* zusammen alle möglichen Typen für die Typ-Vereinbarung zeigt. Dem Leser bereits bekannte Typ-Bezeichner sind *integer*, *boolean*, *char* und *real*.

Der selbstdefinierte Typ BETRIEBSFUNKTION ermöglicht so, nach

TYPE BETRIEBSFUNKTION = (STOP, AUFNAHME, WIEDERGABE, LOESCHEN, VORLAUF, RUECKLAUF);
VAR B1, B2: BETRIEBSFUNKTION;

mit z. B. zwei verschiedenen Variablen B1 und B2 und ihren Werten von STOP bis RUECKLAUF (gewählte Bezeichnung für schnellen Rückwärtslauf) die Steuerung eines Magnetbandgerätes zu programmieren. Eine Vereinbarung der Form
VAR F1,F2,F3: (POSITIV, NEGATIV);
ist dem folgenden hinsichtlich der Variablen F1, F2 und F3 gleichgestellt:

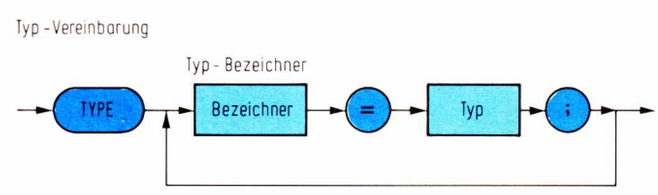


Bild 13. Syntax-Diagramm für die Typ-Vereinbarung

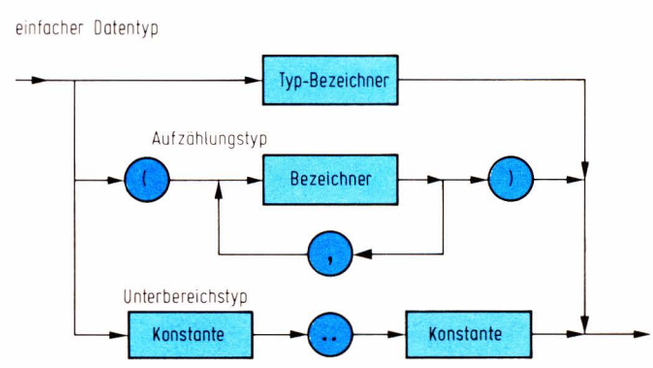


Bild 14. Syntax-Diagramm für den einfachen Datentyp. Eine „Konstante" kann eine ganze Zahl, ein Zeichen oder ein Name einer Konstanten, aber keine reelle Zahl oder eine Zeichenkette sein

TYPE FLANKE = (POSITIV, NEGATIV);
VAR F1, F2, F3: FLANKE;

Es muß also nicht immer ein eigener Typ eingeführt werden. Dies dient mehr der Betonung und Verdeutlichung, ist aber im Zusammenhang mit dem strukturierten Datentypen (s. u.) äußerst praktisch und bei Unterprogrammen zum Teil zwingend vorgeschrieben. Erwähnt sei, daß der Typ *boolean* durch die Aufzählung zweier Werte zustande kommt:

TYPE BOOLEAN = (FALSE, TRUE)

Diese Typ-Vereinbarung (u. a.) geht also jedem PASCAL-Programm voraus, so daß *boolean* standardmäßig zur Verfügung steht.

Eingabe *(input)*

```
4
```

Programm

```
 1 PROGRAM  BUCHSTABENRATEN(INPUT,  OUTPUT);
 2
 3 VAR
 4                GESUCHT,
 5                GEGEBEN:  'A'  ..  'Z'{Unterbereichstyp};
 6                ANTWORT:  BOOLEAN;
 7
 8 BEGIN
 9    GESUCHT  :=  'H';
10    READ(GEGEBEN);
11    ANTWORT  :=  GEGEBEN  <  GESUCHT;
12    WRITE('  EINGEGEBENES  "',  GEGEBEN);
13    WRITELN('"  VOR DEM GESUCHTEN BUCHSTABEN ?',  ANTWORT)
14 END: {BUCHSTABENRATEN}.
```

Ausgabe *(outpout)*

```
PROGRAM  "BUCHSTABEN"  TERMINATED
*  VALUE OUT OF RANGE
*  AT LINE     10


BUCHSTABEN

ANTWORT    =   <UNDEF.>    GEGEBEN    =   <UNDEF.>
GESUCHT    =          H
```

▲ Bild 15. BUCHSTABENRATEN (2. Version). „4" liegt außerhalb des definierten Bereiches und führt so zu einem Laufzeitfehler. Es erfolgt ein Ausdruck des Compilers für die aktuellen Werte der Variablen. Zur besseren Übersichtlichkeit wurden hier die Zeilen durchnummeriert

Die Reihung in der Aufzählung der Werte ordnet den Aufzählungstyp. Daher sind Vergleiche (>, ≧, <, ≦, =) möglich. Von besonderem Nutzen sind schließlich die Standardfunktionen *pred*, *succ* und *ord*. Dabei ist *pred* die sogenannte Vorläufer- und *succ* die Nachfolgerfunktion. Z. B. hat nach der Anweisungsfolge

B1: = STOP;
B1: = SUCC(B1);

die Variable B1 vom Typ BETRIEBSFUNKTION (s. o.) den Wert AUFNAHME. Analoges gilt für beide Funktionen für Argumente vom Typ *integer* und *char* (PRED (78) = 77; SUCC('A') = 'B'); reellwertige Parameter sind dagegen nicht zugelassen. Die Funktion *ord* für die Ordnungszahl liefert eine ganze Zahl für Argumente von geordneten Typen (*integer*, *char*, *boolean*, selbstdefinierte Aufzählungstypen und Unterbereichstypen). Z. B. gilt

ORD(95) = 95, ORD(94) = 94 usw.
ORD(STOP) = 0, ORD(AUFNAHME) = 1 usw.,

aber auch
ORD(FALSE) = 0 und ORD(TRUE) = 1.
Dagegen gilt ORD('A') = 1 nicht für jeden Computer, dies hängt von der jeweiligen Implementation ab. Maschinenintern sind natürlich die Werte eines Aufzählungstyps codiert, was über die Ordnungszahl abgefragt wird. Der Programmierer ist durch diese Eigenschaft von PASCAL davon befreit, die Codierung selbst vorzunehmen und sich die Zuordnungen zu merken (0 sei STOP usw.). In manchen Fällen wird man sich tunlichst an die gewohnte Ordnung (MON bis SON, evtl. SON bis SAM halten). Dennoch muß ein Wermutstropfen im Zusammenhang mit dem Aufzählungstyp verabreicht werden: im Gegensatz zu den Standard-Datentypen (*integer*, *char*, *boolean* und *real*) ist es nicht möglich, Werte dieses Typs mit den Standard-Prozeduren *read*, *readln*, *write*, *writeln* ein- bzw. auszugeben. Dies ist natürlich unbequem und wird

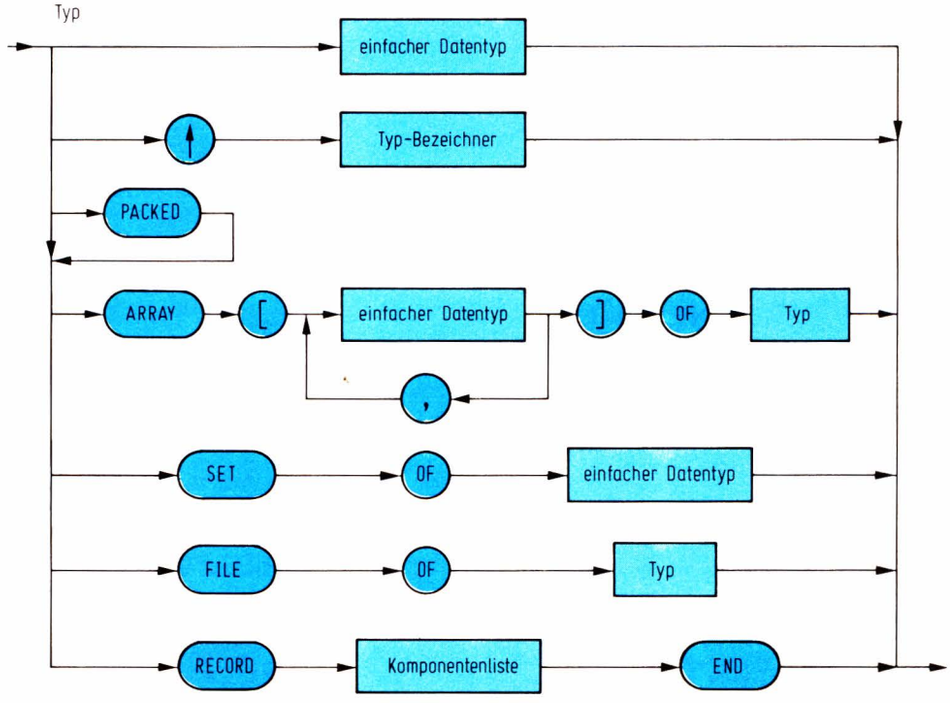


Bild 16. ▶ Syntax-Diagramm für einen Typ. Zu den Einschränkungen für die Bereichsgrenzen eines Feldes (*array*) siehe Abschnitt 4.1

durch die Vielfältigkeit der Aufzählungstypen begründet. Später wird im Abschnitt über die Fallunterscheidung ein Beispiel gebracht und die Lösung dieses Problems mit selbst formulierten Prozeduren aufgezeigt.

Die Unterbereichstypen – auch als Ausschnittstypen bezeichnet – ermöglichen es, den zugelassenen Wertbereich einzuschränken. Dies gilt für alle einfachen Datentypen, außer dem „real"-Typ.

Beispiele für ganze Zahlen:
1901..2099
–50..+50

Im Programmbeispiel BUCHSTABENRATEN erfolgt daher präziser:

VAR GESUCHT, GEGEBEN: 'A'..'Z';

dadurch wird gewährleistet, daß beide Variable nur Werte von Buchstaben haben können. Bei Bereichsüberschreitung erfolgt ein Abbruch des Programms und Anzeige des Fehlers, was von Anlage zu Anlage variiert (*Bild 15*). Eine unzulässige Eingabe sollte in der Regel durch das Programm abgefangen werden. Die Verwendung von Unterbereichstypen bringt auch dann eine zusätzliche Sicherheit, da im weiteren Anweisungsteil zunächst unbemerkte Bereichsüberschreitungen durch fehlerhaftes Programmieren auftreten können. Als Beispiel für einen Unterbereich eines oben angegebenen Aufzählungstyps dienen die Werktage:

VAR WERKTAG: MON..SAM;

Bisher wurden bereits einige Standardnamen erwähnt. Zur besseren Übersicht sind alle wesentlichen Standardnamen in der *Tabelle 2* zusammengefaßt. Erinnert sei hier an die Wortsymbole bzw. an die reservierten Namen der Tabelle 1 (Abschnitt 3.1). Diese Gruppe der Bezeichner hat eine Schlüsselstellung in einem PASCAL-Programm und kann daher nichts anderes bezeichnen. Z. B. kann in der Aufzählung Montag bis Sonntag der Donnerstag nicht mit DO benannt werden, da DO ein Wortsymbol – im Zusammenhang mit den Schleifen – ist, daher: DON (DI, MI usw. wären dagegen möglich). Dem Benutzer steht es aber frei, die *Standardnamen* für eigene Zwecke anders zu verwenden. Sie verlieren dann ihre übliche Bedeutung. Ob das immer sinnvoll ist, ist eine andere Frage. Denkbar wäre es beispielsweise, auf einer Anlage mit großer Bitlänge, den Wertbereich für *integer* mit

TYPE INTEGER = –32767...+32767;

so zu verkleinern, daß das Programm ohne Änderung auch auf einem 16-Bit-Computer mit einfacher Genauigkeit läuft. In einem PASCAL-Programm finden sich daher drei Gruppen von Namen (bzw. Bezeichnern): Wortsymbole, Standardnamen und vom Benutzer gewählte Namen.

4 Zusammengesetzte Datentypen: Felder, Mengen, Dateien und Verbunde

Das Syntax-Diagramm für die zusammengesetzten Datentypen (auch *strukturierte* Datentypen genannt) und für den Zeiger-Typ findet der Leser in *Bild 16*. Im folgenden sollen sie kurz mit Ausnahme des Zeiger-Typs (Abschnitt 7) besprochen werden.

4.1 Felder (array)

Die Felder sind ein klassischer Datentyp und sicher jedem Programmierer vertraut. So sollen nur die Besonderheiten in PASCAL erläutert werden. Der Zugriff auf ein einzelnes Element erfolgt über einen (berechenbaren) Index, der wie in ALGOL in eckigen Klammern steht. Neben ganzzahligen Werten können auch Werte von Zeichen von logischen Werten und von Aufzählungstypen (d. h. von allen geordneten Typen) werden, z. B.:

MESSWERT [1111]
MESSWERT [2*J] mit ganzzahligem J
BFELD ['S']
UMSATZ [MON] usw.;

ZFELD [5.0] hätte eine reelle Größe zum Index und ist nicht erlaubt.

Als eine schwerwiegende Einschränkung in PASCAL muß die Vereinbarung von festen Feldgrenzen angesehen werden, wie z. B.

VAR HAEUFIGKEITSVEKTOR: ARRAY [0..50] OF INTEGER

für ein Feld mit insgesamt 51 Elementen.

Besonders im Falle von Bibliotheksprozeduren ist die starre Festlegung der Feldgrenzen unpraktisch. Diese Vorschrift ist daher sehr umstritten, obwohl sie hinsichtlich der Effektivität des Compilers auch große Vorteile bietet. Viele PASCAL-Übersetzer und die angestrebte Normung enthalten aus diesem Grund die Möglichkeit zur Vereinbarung veränderlicher Feldgrenzen. Abschließend sei dazu gesagt, daß diese enge Vorschrift den Programmierer dazu erzieht, sich genauere Vorstellungen über den Umgang mit Feldern zu machen.

Wie auch die weiteren strukturierten Datentypen lassen sich Felder „packen". Will man oder muß man Speicherplatz sparen, so stellt man dem Wortsymbol *array* das *packed* voran. Ein typisches Beispiel dafür sind Felder von Zeichen, da in der Regel mehrere Zeichen in einem Computerwort dargestellt werden können (in einem 32-Bit-Rechner hätten 4 ASCII-Zeichen Platz). Diese Zeichenketten sind meistens so vereinbart:

CONST N = 10;
TYPE STRING = PACKED ARRAY [1..N] OF CHAR;

Diese relativ kurze Zeichenkette umfaßt genau zehn Zeichen; benötigt man mehr Zeichen, so muß die Konstante N geändert werden. Ebenso bieten sich für die dichtere Informationsdarstellung Felder von logischen Werten (die ja nur 1 Bit benötigen) und von Unterbereichstypen an. Natürlich geht dies auf Kosten der Rechengeschwindigkeit, da der interne Zugriff komplizierter wird.

4.2 Mengen (set)

Mengenoperationen (Vereinigung, Durchschnitt, Differenz, Gleichheit, Ungleichheit, Enthaltensein) lassen zum Teil sehr effektive und elegante Lösungen zu, da sie durch Bitmanipulationen (logisches ODER zweier Bitmuster führt zur Vereinigung usw.) einfach realisiert werden können. Eine Menge in PASCAL wird durch das Umklammern mit eckigen Klammern – die hier eine andere Bedeutung als die Indexklammern bei Feldern haben – dargestellt.

[2, 3, 5, 7]

für die Menge der Primzahlen zwischen 2 und 10;

[U, D, X, E, R, H]

für die Menge aller Adressierungsarten (unmittelbar, direkt, indiziert, erweitert, relativ, inhärent) eines Mikrocomputers. Ein Assemblerbefehl mag dazu z. B. die Menge

[X, E]

umfassen. Führt die Auswertung des Operandenfeldes zu erweiterten Form ([E]), so ergibt der Durchschnitt

[X, E] * [E] = [E].

Da nun die Menge genau ein Element enthält, kann die Adresse aus dem Operanden berechnet werden. Hätte dagegen das Operandenfeld zu [U] geführt, dann wäre das Ergebnis von

[X, E] * [U] = []

die leere Menge [], die in dieser Anwendung eine fehlerhafte Assemblerzeile anzeigt. Ein komplettes Programm mit einer Menge findet der Leser später im Abschnitt 5.2.4 über die Schleifen.

4.3 Dateien (file)

Dateien – häufiger mit File bezeichnet – sind ein wichtiges Mittel für das sequentielle Speichern von

Tabelle 2. Standardnamen der Programmiersprache PASCAL

```
1.  Konstanten
    **********

    FALSE        logischer Wert fuer  "falsch"
    TRUE         logischer Wert fuer  "wahr"
    MAXINT       groesste ganze Zahl                    (anlagenspezifisch)


2.  Einfache Datentypen
    *******************

    INTEGER      ganze Zahl                             -MAXINT .. +MAXINT
    REAL         Gleitkommazahl                         (anlagenspezifisch)
    BOOLEAN      Wahrheitswert                          (FALSE,TRUE)
    CHAR         Zeichen                                (der Zeichenvorrat ist anlagenspezifisch)
    STRING
     oder ALFA   Zeichenkette                           (die Laenge ist anlagenspezifisch)


3.  Programm-Parameter
    ******************

    INPUT        Eingabedatei vom Datentyp TEXT
    OUTPUT       Ausgabedatei vom Datentyp TEXT


4.  Funktionen
    **********

                                                          Argument                  Resultat

    ABS          Absolutbetrag                          INTEGER   REAL          Typ wie Argument
    ARCTAN       Arcustangens                           INTEGER   REAL               REAL
    CHR          Anpassungsoperation                      INTEGER                    CHAR
    COS          Cosinus                                INTEGER   REAL               REAL
    EOF          Abfrage auf das Dateiende                FILE   TEXT              BOOLEAN
    EOLN         Abfrage auf das Zeilenende                 TEXT                   BOOLEAN
    EXP          Exponential-Funktion                   INTEGER   REAL               REAL
    LN           natuerlicher Logarithmus               INTEGER   REAL               REAL
    ODD          Abfrage auf ungeraden Zahlenwert         INTEGER                  BOOLEAN
    ORD          Anpassungsoperation              INTEGER CHAR BOOLEAN Selbstdef.    INTEGER
    PRED         Vorgaenger-Funktion                 INTEGER CHAR Selbstdef.     Typ wie Argument
    ROUND        Rundung                                  REAL                       INTEGER
    SIN          Sinus                                  INTEGER   REAL               REAL
    SQR          Quadrat                                INTEGER   REAL              Typ wie Argument
    SQRT         Wurzel                                 INTEGER   REAL               REAL
    SUCC         Nachfolger-Funktion                 INTEGER CHAR Selbstdef.     Typ wie Argument
    TRUNC        Abrunden                                 REAL                       INTEGER


5.  Prozeduren
    **********

    DISPOSE      Loeschen einer Zeigervariablen
    GET          Lesen aus einer Datei                    FILE TEXT
    NEW          Erzeugen einer Zeigervariablen
    PACK         Packen von Feldern
    PAGE         Seitenvorschub                              TEXT
    PUT          Schreiben in eine Datei                  FILE  TEXT
    READ         Lesen aus einer Datei                       TEXT
    READLN       Lesen aus einer Datei mit Zeilenwechsel     TEXT
    RESET        Ruecksetzen einer Datei zum Lesen        FILE  TEXT    (Ausnahme: INPUT)
    REWRITE      Ruecksetzen einer Datei zum Schreiben    FILE  TEXT    (Ausnahme: OUTPUT)
    UNPACK       Entpacken von Feldern
    WRITE        Schreiben in eine Datei                     TEXT
    WRITELN      Schreiben in eine Datei mit Zeilenwechsel   TEXT
```

Daten. Der Zugriff erfolgt in PASCAL entweder lesend oder schreibend. PASCAL erlaubt es nicht, während des Schreibens auf Lesen oder umgekehrt zu wechseln. Wo dies notwendig ist, kopiert man zunächst die erste Datei (lesend) in eine zweite (schreibend) und fügt an die zweite Datei die weitere Information an. Vor dem ersten Zugriff müssen die Dateien in den Anfangszustand gebracht werden. Dazu existieren die Standardprozeduren

reset (Dateiname) für zu lesende und
rewrite (Dateiname) für zu schreibende Dateien.

Der Zugriff erfolgt dann im allgemeinen Fall mit den Prozeduren

get (Dateiname) und
put (Dateiname),

die später näher besprochen werden.

Die schon benutzten Standardprozeduren *read*, *readln*, *write* und *writeln* sind für das bequeme Arbeiten mit den sogenannten Textdateien geschaffen worden. Eine Textdatei ist als eine Datei mit Elementen vom Zeichentyp zeilenweise strukturiert. Vereinbart ist üblicherweise für die Textdatei der Datentyp TEXT durch TYPE TEXT = FILE OF CHAR;.

(Anmerkung: der Datentyp *char* muß hier um ein Zeichen für den Zeilenwechsel und evtl. Seitenvorschub erweitert sein, das natürlich nicht gedruckt wird, sondern einen Zeilenwechsel vornimmt.) Verwendet man die Textdateien *input* und *output*, so braucht der Programmierer erstens den Dateinamen in den E-/A-Standardprozeduren nicht angeben. (WRITE (ANTWORT) und WRITE (OUTPUT,ANTWORT) sind z. B. gleichwertig). Zweitens sind diese beiden Dateien automatisch initialisiert, so daß reset und rewrite entfallen. *input* und *output* sind durch die Variablen-Vereinbarung

VAR INPUT,OUTPUT : TEXT;

vordefiniert. Während

FILE OF INTEGER;
FILE OF STRING;
FILE OF RECORD...END

mögliche Konstruktionen von Dateien sind, ist bei vielen Compilern das syntaktisch richtige

FILE OF FILE OF....

(*Bild 16*) verboten.

Mit den Standardfunktionen *eof* und *eoln* läßt sich zu jedem Zeitpunkt abfragen, ob das Dateiende (bzw. Zeilenende, daher ist *eoln* nur auf Textdateien anwendbar) erreicht ist. Dies bietet eine einfache Möglichkeit der Programmierung, z. B. Daten so lange einzulesen, bis die Datei erschöpft ist. Dabei spielt es keine Rolle, ob keine oder viele Daten vorhanden sind. Auch hier brauchen die Testdateien *input* und *output* nicht eigens als Parameter zu erscheinen.

4.4 *Verbunde (record)*

Verbunde – von *N. Wirth* mit Datensatztyp bezeichnet – ermöglichen es, Komponenten verschiedener Herkunft zu einer Struktur zusammenzuführen (zu verbinden). Als Beispiele seien genannt:

komplexe Zahl:

```
RECORD      RE : REAL;
            IM : REAL
END;
```

(vor dem END braucht kein Strichpunkt stehen)

Datum:

```
RECORD      TAG : 1..31;
            MONAT : 1..12;
            JAHR : 1901..2099
END;
```

Karteiblatt eines Foto-Archivs:

```
RECORD      OBJEKT : STRING;
            AUFNAHME : DATUM;
            UHRZEIT : ....;
            BILDNR : INTEGER;
            BLENDE : REAL;
            ENTFERNUNG : REAL;
            FILM : (FIRMA1,...)
END;
```

Das letzte Beispiel zeigt, daß in einem Verbund ein weiterer (hier ein Verbund für das Datum) stehen darf.

Eine Kartei kann entweder als ein Feld (ARRAY [1..N] OF RECORD ..... END), als Datei (FILE OF ...) oder durch eine mit Zeigern verkettete Struktur konstruiert werden. Im letzteren Fall verweist ein „Karteiblatt" auf ein weiteres.

Das nächste Programmbeispiel berechnet das Produkt zweier komplexer Zahlen (*Bild 17*). Nach dem Einlesen (abwechselnd Real- und Imaginärteil für jede Zahl) erfolgt die Berechnung des Produktes und anschließend die Ausgabe, wobei auf eine komfortablere

Eingabe *(input)*

```
1.1  0.4  0.5  2.0
```

Programm

```
PROGRAM  KOMPLEXEMULTIPLIKATION(INPUT,  OUTPUT);

TYPE
            KOMPLEX  =  RECORD
                                RE:  REAL;
                                IM:  REAL
                        END;

VAR
              PRODUKT,
                   Z1,
                   Z2:  KOMPLEX;

BEGIN
   READ(Z1.RE,  Z1.IM,  Z2.RE,  Z2.IM);
   PRODUKT.RE  :=  Z1.RE  *  Z2.RE  -  Z1.IM  *  Z2.IM;
   PRODUKT.IM  :=  Z1.IM  *  Z2.RE  +  Z1.RE  *  Z2.IM;
   WRITELN('   REALTEIL      IMAGINAERTEIL');
   WRITELN(PRODUKT.RE:  12,  '      ',  PRODUKT.IM:  12)
END  {KOMPLEXEMULTIPLIKATION}.
```

Ausgabe *(output)*

```
  REALTEIL       IMAGINAERTEIL
-2.500E-001       2.400E+000
```

Bild 17. Programm KOMPLEXE MULTIPLIKATION. $Z_1 = 1{,}1 + j0{,}4$, $Z_2 = 0{,}5 + j2$ werden eingegeben und führen zu dem ausgedruckten Ergebnis. Der Zugriff auf eine Komponente (RE oder IM) erfolgt durch das Anhängen eines Punktes an den Variablennamen (Z 1, Z 2, PRODUKT)

Darstellung verzichtet wurde. Der Zugriff auf eine Komponente der Variablen vom Typ KOMPLEX erfolgt über das Anhängen eines Punktes an den Variablennamen und den Bezeichner für die Komponente (RE oder IM im Beispiel des Verbundes KOMPLEX).

In manchen Fällen hängt die Struktur eines Verbundes von den Ausprägungen der Komponenten ab. Die Darstellung komplexer Zahlen kann nicht nur in kartesischen, sondern auch in Polar-Koordinaten erfolgen. Nach den Vereinbarungen

```
TYPE KOMPLEX2 = RECORD
                CASE KOORD : CHAR OF
                'K' : (RE,IM : REAL);
                'P' : (BETRAG,WINKEL : REAL)
                END;
VAR C: KOMPLEX2;
```

kann nach der Anweisung C.KOORD := 'P' auf Betrag und Phase durch C.BETRAG und C.WINKEL zugegriffen werden. Diese Form eines Verbundes bedarf einiger Aufmerksamkeit. Es wird dem Leser daher zu diesem Punkt das Studium eines Lehrbuches empfohlen.

In der ELEKTRONIK wurde kürzlich ein Komfort-Programm zum Rechnen mit komplexeren Zahlen veröffentlicht |11|. Die dort gewählte Darstellung läßt sich in PASCAL wie folgt vereinbaren:

```
TYPE KOMPLEX4 = RECORD
                RE,
                IM,
                BE,
                WI : REAL
                END;
VAR X, Y, Z, Q : KOMPLEX4;
```

Jede komplexe Zahl führt so vier Komponenten (Real-, Imaginärteil, Betrag und Winkel) mit sich. Wie in jenem Aufsatz gezeigt, bestimmen bis auf einen Sonderfall zwei dieser Komponenten eine Zahl. Die Operationen werden zweckmäßigerweise als Prozeduren formuliert.

In *Bild 18* und *19* sind die Syntax-Diagramme für den Verbundtyp und zur Vervollständigung für die Variable zu finden.

5 Anweisungen

Von den Möglichkeiten, die PASCAL für die Steuerung des Programmflusses bietet, kennt der Leser bereits die Wertzuweisung, die Prozeduranweisung und die (kleine, aber nicht bedeutungslose) leere Anweisung. Für die graphischen Mittel, die Aktionen eines Programms darzustellen, sei auf den ELEKTRONIK-Aufsatz |14| verwiesen. Dort finden sich die schon länger eingeführten Flußdiagramme und die neueren Struktogramme. Die Syntax-Diagramme zu diesem Abschnitt befinden sich in *Bild 20a* (Anweisung) und *20b* (strukturierte Anweisung); das umfangreiche Diagramm für einen *Ausdruck* zeigt *Bild 21*.

5.1 Einfache Anweisungen

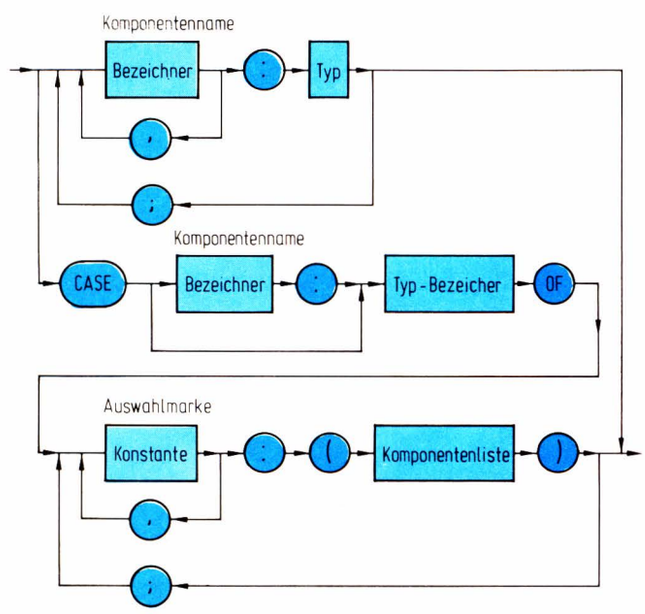
5.1.1 Wertzuweisung

In den bisherigen Programmbeispielen findet der Leser die Wertzuweisungen

ZEIT := 15.0	für die reelle Variable ZEIT,
GESUCHT := 'H'	für die Zeichen-Variable GESUCHT.

Wie bei anderen mit ALGOL verwandten Sprachen gibt es den sogenannten Zuweisungsoperator („:=“), der vom Gleichheitszeichen („=“) zu unterscheiden ist. Auf der rechten Seite steht im allgemeinen ein Ausdruck.

Auf die Vielfalt eines Ausdruckes kann hier leider nicht näher eingegangen werden. Dem Leser wird empfohlen, sich an Hand der bereits erwähnten Lehrbücher näher zu informieren. In den folgenden Beispielen sind einige wichtige Ausdrücke enthalten und werden dort besprochen.



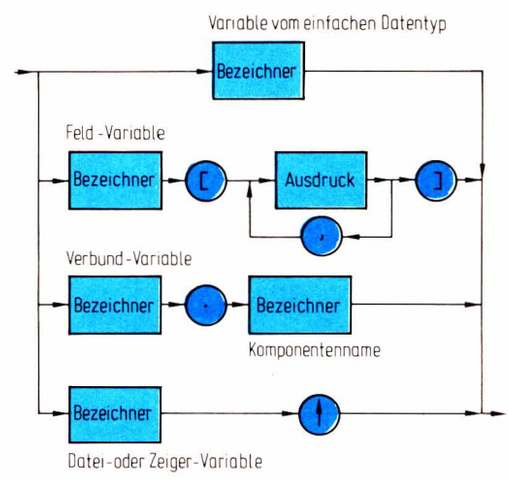


Bild 19. Syntax-Diagramm für eine Variable

◀ Bild 18. Syntax-Diagramm für die Komponentenliste eines Verbundes. Der mit dem Wortsymbol *case* eingeleitete Teil beschreibt den sogenannten Variantenteil

Die höchste Priorität hat der Operator *not*, dann folgen die Multiplikationsoperatoren (, /, *div*, *mod*, *and*), dann die Additionsoperatoren (+, –, *or*) und schließlich die Vergleichsoperatoren (=, <>, <, <=, >=, >, *in*). Durch Klammerung läßt sich wie üblich eine andere Reihenfolge der Auswertung des Ausdrucks erzwingen. Wie man sieht, existieren zwei Divisionszeichen: *div* gilt für ganzzahlige Operanden und führt zu einem ganzzahligen Resultat (z. B. ergibt 19 *div* 5 den Wert 3, der Rest 4 wird weggeworfen). Mit dem anderen Zeichen („/") erhält man immer eine reelle Zahl als Ergebnis – auch bei Operanden vom Typ *integer*. Ist M als eine „integer"-Größe vereinbart (Y *real* oder *integer*), so ist der Ausdruck

M := Y/56

syntaktisch falsch, da rechts ein reelles Resultat auftritt, das einer ganzzahligen Variablen zugewiesen werden soll. Andere Sprachen lösen dieses Problem durch automatische Rundung. PASCAL ist hier (und anderswo) ein strenger Lehrmeister, der den Programmierer zwingt, die gewählten Datentypen streng einzuhalten. Der Ausdruck lautet dann richtig

M := *round* (Y/56) oder
M := *trunc* (Y/56)

falls entweder gerundet oder abgerundet werden soll.

5.1.2 Prozeduranweisung

Die Prozeduranweisung – ähnlich dem CALL-Aufruf in FORTRAN und BASIC – dient zum Aufruf einer Folge von Anweisungen, die im vorderen Teil des Programms vereinbart wurden. Der Aufruf erfolgt durch die Nennung des Prozedurnamens, dem u. U. Parameter in runden Klammern folgen. Wie jede andere Anweisung wird sie von der nächsten durch ein „;" getrennt. Die Prozeduranweisung ist von einem Funktionsaufruf verschieden; letzteres liefert einen Wert für einen einfachen Datentyp. Dem Leser sind in Gestalt der Standardprozeduren WRITE und READ bereits Beispiele für Prozeduranweisungen begegnet. Wegen ihrer Wichtigkeit werden Prozeduren und Funktionen in Abschnitt 6 ausführlich beschrieben.

5.1.3 Sprunganweisung

Auch PASCAL kennt den unbedingten Sprung mit *goto*. Die Existenz einer Sprunganweisung ist natürlich umstritten, da sie oft dem erklärten Ziel, Programme übersichtlich zu gestalten, zuwiderläuft. So ist die Anwendung von *goto* in PASCAL durch harte Bestimmungen erschwert. Sprungziele sind die sogenannten Marken, für die nur vorzeichenlose ganze Zahlen (meist zwischen 0 und 9999) zugelassen sind. Sie müssen im Marken-Vereinbarungsteil (ohne Syntax-Diagramm) ausdrücklich definiert werden. Aus einem Block (z. B. Prozedur) darf herausgesprungen, nicht aber hineingesprungen werden. Es sei empfohlen, die Sprunganweisung nur selten zu verwenden (daher hier auch kein Programmbeispiel).

5.1.4 Leere Anweisung

Die leere Anweisung ist eine syntaktische Hilfskonstruktion, um auch das Aufeinanderfolgen von ...; ;... oder von ...; *end* ... in einem PASCAL-Programm zuzulassen. Letzteres tritt dann auf, wenn der Programmierer automatisch hinter jeder Anweisung ein „;" setzt. Ähnlich wie ein Kommentar hat die leere Anweisung während des Programmlaufes keine Wirkung.

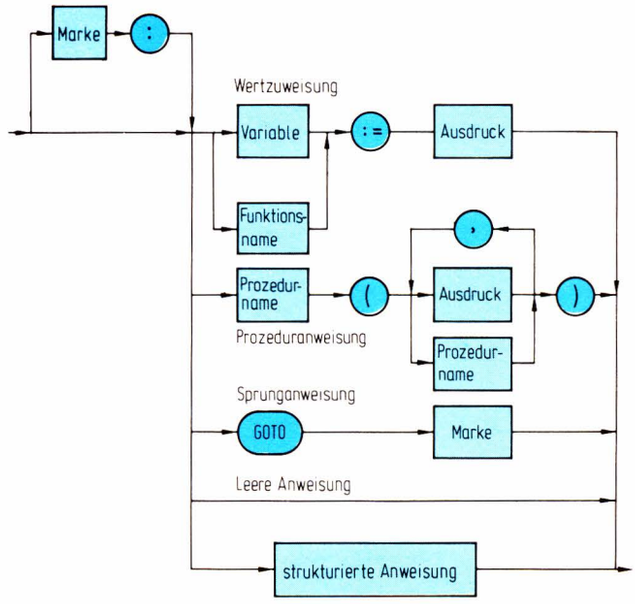


Bild 20a. Syntax-Diagramm für die Anweisung. Zur Vervollständigung ist der Pfad mit der Marke eingezeichnet

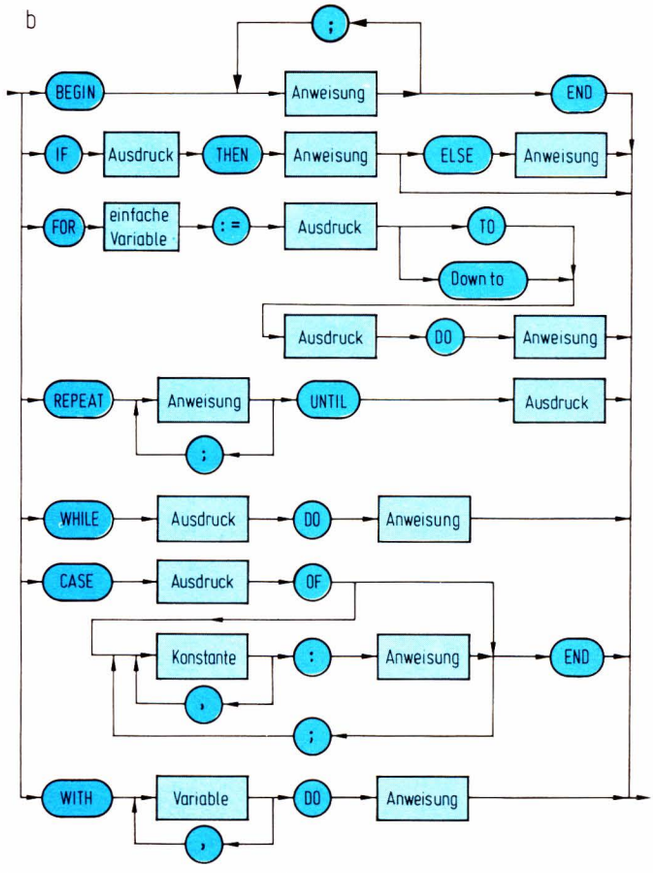


Bild 20b. Syntax-Diagramm für die strukturierte Anweisung

5.2 *Strukturierte Anweisungen*

5.2.1 Anweisungssequenz

Eine Folge von Anweisungen, die eine Einheit bilden soll, wird mit *begin* und *end* eingeklammert. Diese Zusammenfassung tritt oft bei Schleifen auf und wird später in den Programmbeispielen häufig vorkommen. Nach außen wirkt die Anweisungssequenz wie *eine* Anweisung. In der Literatur wird sie auch mit Verbundanweisung bezeichnet. Um hier Verwechslungen mit den Verbunden („record"-Typen) zu vermeiden, wurde die Bezeichnung wie in [6] gewählt.

5.2.2 Bedingte Verzweigung

Die von einer Bedingung abhängige Verzweigung in einer Anweisungsfolge ist ein klassisches Programmierelement. PASCAL kennt die einseitige und die zweiseitige Verzweigung, die mit dem Wortsymbol *if* eingeleitet werden. Nach einem logischen Ausdruck, der ausgewertet zu den Konstanten TRUE oder FALSE führt, erfolgt nach *then* eine Anweisung. Im Falle der einseitigen bedingten Verzweigung wird dieser Teil ausgeführt, wenn TRUE zutraf. Falls FALSE zutraf, wird er nicht ausgeführt und sofort mit der nächsten Anweisung der Programmablauf fortgesetzt. Bei der zweiseitigen Verzweigung wird im Unterschied dazu der Programmteil nach dem Wortsymbol *else* abgearbeitet. Hier sei auf die wichtige syntaktische Regel hingewiesen, daß vor einem *else* kein „;" stehen darf.

Als Beispiel möge nochmals die Berechnung der Übergangsfunktion dienen, für die im ersten Teil bereits ein Programm angegeben wurde (Bild 11c). Bei seiner Benutzung mußte beachtet werden, daß nur Werte für $t \geqq 0$ eingegeben wurden. Negative Werte führten zu einem Berechnungsfehler, da für ein Verzögerungsglied erster Ordnung die Übergangsfunktion

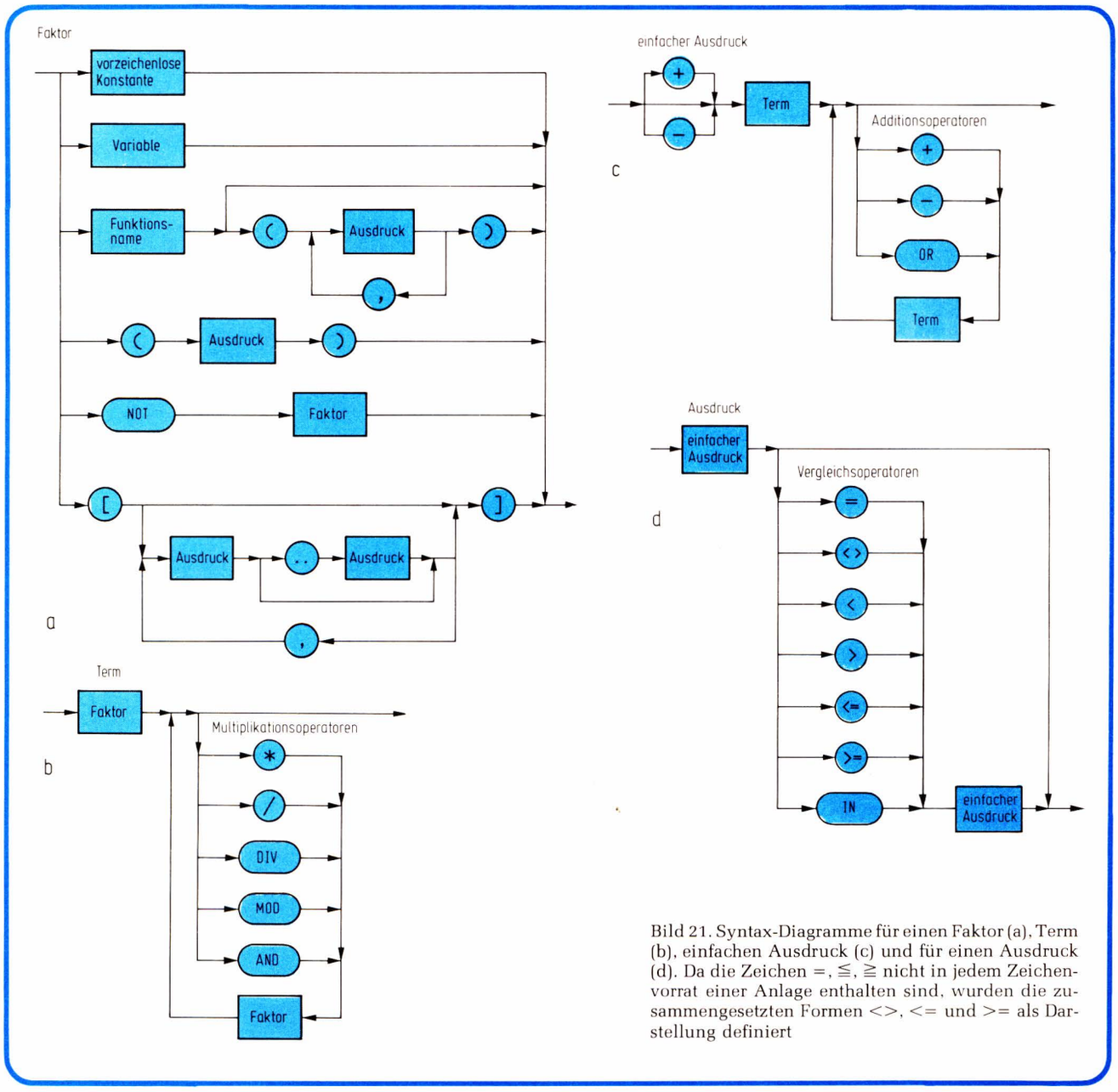


Bild 21. Syntax-Diagramme für einen Faktor (a), Term (b), einfachen Ausdruck (c) und für einen Ausdruck (d). Da die Zeichen =, ≦, ≧ nicht in jedem Zeichenvorrat einer Anlage enthalten sind, wurden die zusammengesetzten Formen <>, <= und >= als Darstellung definiert

$$y = \begin{cases} 1-e^{-t/\tau} & t \geqq 0 \\ 0 & t<0 \end{cases}$$

gilt. Die zweiseitige bedingte Verzweigung ist dafür die korrekte Form der Ablaufsteuerung. Das so geänderte Programm VERZOEGERUNGSGLIED zeigt *Bild 22*. In ihrer Wirkung diesem Programm gleichwertig wäre auch folgendes Teilstück mit der einseitigen bedingten Verzweigung:

```
READ (ZEIT);
IF ZEIT<0.0 THEN ZEIT:=0.0;
AUSGANG: = 1.0 – EXP(-ZEIT/ZEITKONSTANTE);
```

Hier wird durch die „if"-Anweisung (in einer Reihe von Literaturstellen werden die verschiedenen Anweisungsformen mit dem entsprechenden Wortsymbol bezeichnet) sichergestellt, daß positive Werte und Null für die Zeit in die Berechnung eingehen. Der Programmierstil in Bild 22 ist jedoch klarer und eindeutiger; er ist daher vorzuziehen.

5.2.3 Schleifen

5.2.3.1 Laufanweisung

Schleifen dienen zur wiederholten Ausführung von einer oder mehreren Anweisungen. Ist die Anzahl der Durchläufe bekannt, so wird man dazu in der Regel die Laufanweisung heranziehen. Wenn nicht, zieht man eine der beiden bedingten Schleifen – „repeat"-Anweisung, „while"-Anweisung – heran. Eine Lösung mit *goto* ist in PASCAL indiskutabel.

Charakteristisch für die Laufanweisung (bzw. „for"-Anweisung) ist, daß die Laufvariable eine vereinbarte Variable eines einfachen Datentyps mit Ausnahme des „real"-Typs sein kann. Das Verbot von

Eingabe *(input)*

```
-5.0
```

Programm

```
PROGRAM  VERZOEGERUNGSGLIED(INPUT,  OUTPUT);
   {   REAKTION AUF EINEN EINHEITS-SPRUNG }

CONST
        ZEITKONSTANTE  =  5.0  { SEC };

VAR
                    ZEIT,
                AUSGANG:  REAL;

BEGIN
   READ(ZEIT);
   IF   ZEIT   >=  0.0
   THEN
      AUSGANG   :=  1.0  -  EXP(-  ZEIT  /  ZEITKONSTANTE)
   ELSE
      AUSGANG   :=  0.0;
   WRITELN('      ZEIT=              ',  ZEIT:  12:  4);
   WRITELN('      AUSGANGSGROESSE=',  AUSGANG:  12:  4)
END  {VERZOEGERUNGSGLIED}.
```

Ausgabe *(output)*

```
ZEIT=                  -5.0000
AUSGANGSGROESSE=             0
```

Bild 22. Programm VERZOEGERUNGSGLIED. Diese Version berücksichtigt auch negative Zahlenwerte für die Zeit

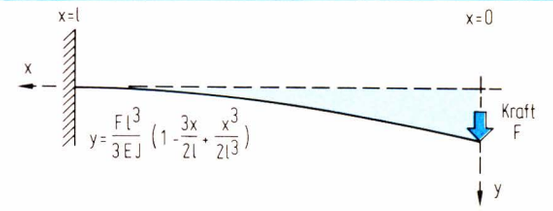


Bild 23. Einseitig eingespannter Balken mit punktförmiger Belastung in x = 0

„real"-Größen hat natürlich Widerspruch hervorgerufen. Diese Beschränkung wurde eingeführt, weil die statistisch am häufigsten verwendete Schrittweite 1 ist und die Laufanweisung dadurch einfacher wird. Das ist weithin akzeptiert. Auf der anderen Seite ist es aber in PASCAL möglich, die Laufanweisung mit anderen als „integer"-Variablen zu formulieren, z. B.:

```
FOR   ZEICHEN:='A' TO 'Z' DO... ;
FOR B1:=STOP TO VORLAUF DO... ;
```

Dabei sei ZEICHEN vom Datentyp *char* und B1 von dem oben definierten Aufzählungstyp BETRIEBSFUNKTION.

Im ersten Programmbeispiel mit der „for"-Anweisung soll eine Tabelle für die Biegelinie eines einseitig

Eingabe *(input)*

```
10  2.06E+11  0.4  0.03
```

Programm

```
PROGRAM  BIEGELINIE(INPUT,  OUTPUT);

CONST
                MUEMAX  =  10;
                    PI  =  3.14159265359;
                 YSKAL  =  1.0E+6  { Y in Mikrometer };

VAR
                      L,   { Laenge des Balkens }
                      F,   { Kraft }
                      E,   { Elastizitaetsmodul }
                      J,   { Traegheitsmoment }
                      D,   { Durchmesser }
                      X,
                  DELTA,   { Schrittweite fuer "x" }
                      Y,
                 FAKTOR:  REAL;
                    MUE:  INTEGER;

BEGIN
   { Eingabe der mechanischen Groessen in den
        Grundeinheiten Newton und Meter }
   READ(F,  E,  L,  D);
   J  :=  D  *  D  *  D  *  D  *  PI  /  64.0;
   FAKTOR  :=  F  *  L  *  L  *  L  /  (3.0  *  E  *  J);
   DELTA  :=  L  /  MUEMAX;
   WRITELN('        NR.          X            Y'); :
   WRITELN('                METER      MIKROMETER');
   FOR  MUE  :=  0  TO  MUEMAX  DO
      BEGIN
         X  :=  DELTA  *  MUE;
         Y  :=  YSKAL  *  FAKTOR  *  (1.0  -  1.5  *  X  /  L
              +  0.5  *  X  *  X  *  X  /  (L  *  L  *  L));
         WRITELN(MUE,  X:  12:  3,  Y:  12:  2)
      END  {FOR}
END  {BIEGELINIE}.
```

Ausgabe *(output)*

```
      NR.          X            Y
                METER     MIKROMETER
       0            0        26.05
       1        0.040        22.15
       2        0.080        18.34
       3        0.120        14.68
       4        0.160        11.25
       5        0.200         8.14
       6        0.240         5.42
       7        0.280         3.16
       8        0.320         1.46
       9        0.360         0.38
      10        0.400            0
```

Bild 24. Programm BIEGELINIE mit Eingabe und Ausgabe der Biegung y in Abhängigkeit vom x

eingespannten Trägers erstellt werden. Als Erläuterung diene *Bild 23* mit der Gleichung für die Biegelinie [aus 15]. Das Programm BIEGELINIE (*Bild 24*) berechnet nach der Eingabe der Größen für die Kraft (F=10 N), für den Elastizitätsmodul von Stahl St 38 und für Länge und Durchmesser des Trägers (l = 0,4 m, D = 0,03 m) das Trägheitsmoment J, den Faktor $F \cdot l^3/3EJ$ und die reelle Schrittweite Δx. Die Laufvariable ist ganzzahlig. x berechnet sich dann in der Anweisungssequenz aus $\Delta x \cdot \mu$, so daß x von 0 bis x=l läuft. Die Berechnung von y erfordert einen Ausdruck, der zwei Zeilen umfaßt.

Durch die Ausgabe-Prozedur WRITELN werden zeilenweise die Schrittnummer μ, x und y formatiert ausgegeben. Zu beachten ist die Richtung von x und y nach Bild 23. Dieses Beispiel macht auch deutlich, daß mit großen und kleinen Buchstaben für Bezeichner eine deutlichere Namensgebung möglich wäre (z. B. statt D für den Durchmesser). Eine Erweiterung des Programmes für verschiedene Querschnitte wäre am besten durch einen Verbundtyp für den Balken beschreibbar:

```
TYPE
  FLAECHE=(KREIS,RECHTECK,...);
  BALKEN =RECORD
            LAENGE : REAL;
            CASE FORM : FLAECHE OF
               KREIS : (D : REAL);
               RECHTECK : (SEITE 1,
               SEITE 2 : REAL);
               :
               END {BALKEN};
```

In dem Verbund BALKEN (siehe dazu Bild 18) leitet das Wortsymbol *case* den sogenannten Variantenteil ein. Die Pünktchen zeigen an, daß hier weitere Flächenformen definiert werden können. Entsprechend dieser Datenstruktur wird dann im Anweisungsteil das jeweilige Flächenträgheitsmoment mit der noch zu besprechenden Auswahlanweisung (5.2.4) bestimmt.

Das nächste Beispiel demonstriert wieder die Laufanweisung zur Erzeugung einer binären Pseudo-Zufallsfolge. Pseudo-Rauschsignale aus einer Anordnung, wie in *Bild 25* gezeigt ist, werden mit Vorteil zur Identifikation dynamischer Systeme eingesetzt. Aus der reichhaltigen Literatur sei [16] herausgegriffen. Das Programm PSEUDOZUFALLSFOLGE (*Bild 26*) wird dazu benutzt, die elektronisch erzeugten Rauschsignale (Eingangssignal des experimentell untersuchten Systems) bei der Auswertung zu simulieren. Dies erspart die Aufzeichnung und Speicherung eines Kanals. Die Pseudo-Zufallsfolge ist periodisch. Ihr Frequenzspektrum entspricht mit guter Näherung einem weißen Rauschen, so daß viele Frequenzkomponenten das dynamische System anregen.

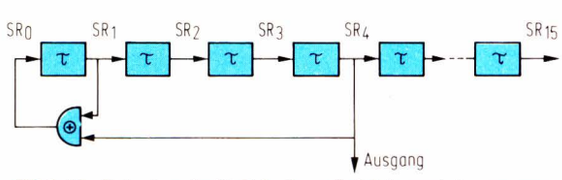


Bild 25. Prinzipschaltbild eines Schieberegistergenerators mit den Verzögerungsgliedern τ

Ein Generator für eine Pseudo-Impulsfolge besteht aus einem mehrstufigen Schieberegister, von dem zwei Abgriffe zu einem EXOR-Gatter führen. Der Ausgang dieses Gatters wird mit dem ersten Register verbunden. Durch Verschiebeimpulse erhält man ein periodisches pseudozufälliges Signal an jedem Ausgang von der Länge (Periode)

$$P = 2^N - 1$$

(N : Anzahl der Stufen). Im Programm PSEUDOZUFALLSFOLGE wird das Schieberegister durch ein Feld von logischen Werten dargestellt. Die Rückführung erfolgt von der N-ten Stufe und von ABGRIFF. Der Ausgang des exklusiven ODER wird zunächst in der nullten Stufe gespeichert. Die Verschiebung geschieht durch wiederholtes Umspeichern der Feldelemente. Dazu wird die zweite Form der „for"-Anweisung mit dem Abwärtszählen herangezogen. Im Programm sieht man Laufanweisungen mit einer einzelnen Anweisung und mit einer Anweisungssequenz (*begin ... end*) nach dem *do*-Symbol.

5.2.3.2 „repeat"-Anweisung

An Hand der bedingten Schleife läßt sich die iterative Lösung einer transzendenten Gleichung demonstrieren. Seit langem kennt man in der Astronomie die sogenannte Kepler-Gleichung

$$M = E - e \sin E$$

(nicht zu verwechseln mit den Keplerschen Gesetzen). Sie wird bei der Berechnung von Planeten-(bzw. Satelliten-)Bahnen benutzt. Eine geometrische Veranschaulichung bringt *Bild 27*. Die mittlere Exzentrizität M ist der Winkel für den auf einer Kreisbahn mit gleichmäßiger Geschwindigkeit umlaufenden gedachten Himmelskörper. Dieser ist für den Zeitpunkt der Bahnbestimmung gegeben (M = 0,5 rad = 28,64789°); ebenso die numerische Exzentrizität mit e = 0,2. Zunächst wird die Kepler-Gleichung in die Gestalt

$$\varphi(E) = E$$

umgeformt und rechts E eingesetzt

$$\varphi(E) = M + e \cdot \sin E.$$

Beginnend mit einem Startwert für die exzentrische Anomalie E führt jede Berechnung zu $\varphi(E)$, der als neues E für den nächsten Schritt dient. Die Berechnung wird abgebrochen, wenn die Differenz zwischen den alten und neuen E eine vorgegebene Schranke unterschreitet. Die im Programm KEPLERGLEICHUNG (*Bild 28*) enthaltene Anweisungsfolge zwischen *repeat* und *until* wird so lange ausgeführt, *bis* die Bedingung $\Delta < \varepsilon$ den Wert TRUE erhält. Dann – und nur dann! – wird die Schleife verlassen und mit den ab-

schließenden Ausgabeanweisungen fortgefahren. Die Tabelle in Bild 28 enthält die Schriftnummer, den Wert für E und die Differenz Δ. Die Sinusfunktion verlangt als Argument den Winkel in Radiant-Einheiten; für E erhält man dann die Lösung E = 0,61546817 rad = 35,26450°.

Fehlerquellen beim Programmieren von Schleifen sind oft falsch gewählte Vorbedingungen. In diesem Beispiel sind dies die Größen GROSSEE und SCHRITTNR, die vor „repeat" Werte erhalten müssen. Für die vorliegende Iteration sind die Konvergenzbedingungen [z. B. in 17] gut untersucht, so daß eine Beendigung sicher erfolgt, wenn ε nicht zu klein gewählt worden ist (abhängig von der Stellenzahl für die „real"-Größen). In [17] sind auch schnellere Iterationsverfahren – wie das Verfahren nach *Newton* – ausführlich beschrieben.

Eingabe *(input)*

```
4    1
```

Programm

```
PROGRAM  PSEUDOZUFALLSFOLGE(INPUT,  OUTPUT);

CONST
          MAXSTUFEN  =  15;

VAR
                  MUE,
              ABGRIFF,
                    N:  1..  MAXSTUFEN;
                 JOTA,
              PERIODE:  INTEGER;
                   SR:  ARRAY  [0..MAXSTUFEN]OF  BOOLEAN;

BEGIN
   READ(N,  ABGRIFF);
   { Startzustand des Schieberegisters setzen }
   FOR  MUE  :=  1  TO  N  DO
      SR[MUE]  :=  TRUE;
   { Berechnung der Periodenlaenge (=N**2-1) }
   PERIODE  :=  1;
   FOR  MUE  :=  1  TO  N  DO
      PERIODE  :=  PERIODE  *  2;
   PERIODE  :=  PERIODE  -  1;
   { Berechnung der binaeren Zufallsfolge }
   FOR  JOTA  :=  1  TO  PERIODE  DO
      BEGIN
         WRITELN(JOTA,  SR[N],  ORD(SR[N]));
         { exklusives ODER }
         SR[0]  :=  (SR[N]  AND  NOT  SR[ABGRIFF])  OR  (NOT
            SR[N]  AND  SR[ABGRIFF]);
         { Verschiebung }
         FOR  MUE  :=  N  DOWNTO  1  DO
            SR[MUE]  :=  SR[MUE  -  1]
      END  {FOR}
END  {PSEUDOZUFALLSFOLGE}.
```

Ausgabe *(output)*

```
 1     TRUE     1
 2     TRUE     1
 3     TRUE     1
 4     TRUE     1
 5    FALSE     0
 6     TRUE     1
 7    FALSE     0
 8     TRUE     1
 9     TRUE     1
10    FALSE     0
11    FALSE     0
12     TRUE     1
13    FALSE     0
14    FALSE     0
15    FALSE     0
```

Bild 26. Programm PSEUDOZUFALLSFOLGE. Berechnung für ein vierstufiges Schieberegister (Name: SR). Die Tabelle enthält die Schrittnummer, den Inhalt der 4. Stufe (TRUE oder FALSE) und die Ordnungszahl davon. Letztere bekommt man durch Anwendung der Standardfunktion *ord* auf das Feldelement SR[N]

5.2.3.3 „while"-Anweisung

Im Unterschied zur „repeat"-Anweisung wird hier die Bedingung zuerst geprüft und im Falle von TRUE die Anweisung nach *do* ausgeführt. Bei FALSE wird die „while"-Anweisung verlassen. Während also die Anweisungsfolge zwischen *repeat* und *until* mindestens einmal durchlaufen wird, kann es vorkommen, daß das Innere der Schleife *while...do...* überhaupt nicht erreicht wird. Eine elegante Möglichkeit für die Anwendung ist die Standardfunktion *eof*, die das Dateiende von *input* abfragt. Sie liefert einen logischen Wert, je nachdem, ob sich in der Textdatei *input* noch weitere Daten befinden. Ist das Dateiende erreicht, so gilt TRUE, sonst FALSE. Durch den Operator *not* erhält man die passende Bedingung für die „while"-Anweisung im Programm ABZAEHLEN (*Bild 29*). Das Programm liest Zeichen für Zeichen ein und prüft, ob es sich um einen Vokal handelt. Dazu wird eine Menge mit dem Namen VOKAL konstruiert, die die fünf Vokale des Alphabets enthält. Die Konstruktion erfolgt durch Aufzählen und Einklammern der Zeichen 'A'..'U' und durch eine Wertzuweisung an die Variable VOKAL. Sie mußte natürlich als eine Menge von den Buchstaben des Alphabetes vereinbart sein. Aus anlagenspezifischen Einschränkungen (eine Menge darf nur 59 Elemente umfassen) konnte die sinnvollere Vereinbarung

VOKAL : SET OF CHAR;

nicht gewählt werden. Um auch Satzzeichen usw. mit READ(ZEICHEN) eingeben zu können, ist ZEICHEN vom einfachen Datentypen *char*. In der bedingten Verzweigung wird durch den Ausdruck

ZEICHEN IN VOKAL

geprüft, ob das zuletzt eingegebene Zeichen als Element in der gebildeten Menge VOKAL enthalten ist

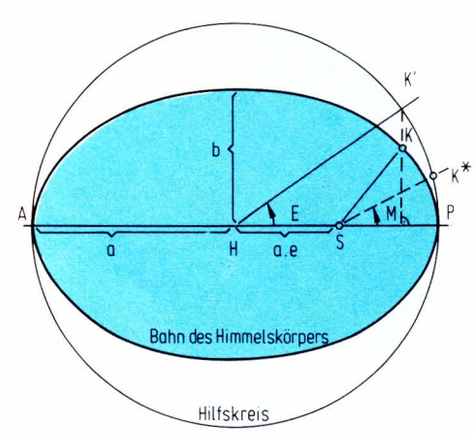


Bild 27. Geometrische Veranschaulichung der Kepler-Gleichung M=E-e sin E. Der Himmelskörper K läuft (ungleichmäßig) um den Brennpunkt S (z. B. die Sonne) auf einer elliptischen Bahn (Halbachsen a und b). K* wäre der hypothetische Ort, wenn er sich mit einer gleichmäßigen mittleren Geschwindigkeit um H bewegen würde

(TRUE) oder nicht (FALSE). Als Eingabe für das übersetzte und geladene ABZAEHLEN wurde der Text des Programms selbst gewählt. Während die Vorbedingung für die Variable ZAEHLER durch den Programmierer (ZAEHLER : = 0;) erfolgen muß, wird die Vorbedingung für *eof* durch den Compiler vorgenommen. Dies geschieht automatisch für die Datei *input* nach der Angabe als Parameter im Programmkopf zu Beginn des Programmlaufes.

5.2.4 Auswahl-Anweisung

Eine Reihe von Problemlösungen verlangt die Entscheidung zwischen mehr als zwei Alternativen. Dafür existiert in PASCAL eine eigene Ablaufstruktur: die Auswahl-Anweisung mit dem einleitenden Wortsymbol *case*. Als Demonstration diene erneut das Raten eines Buchstabens. Die im Abschnitt 3.4 vorgestellten Programme (Bild 12 bzw. 15) sind sicherlich nicht zufriedenstellend. Das liegt daran, daß sie nur eine Abfrage enthalten und kein wiederholtes Raten bis zum Finden des Buchstabens erlauben.

Eingabe *(input)*

```
0.2 0.5
```

Programm

```
PROGRAM  KEPLERGLEICHUNG(INPUT,  OUTPUT);
{ Iterationsverfahren zur Loesung der Kepler-Gleichung }

CONST
              EPSILON  =  1E-11;

VAR
              SCHRITTNR:  INTEGER;
                KLEINEE,
                GROSSEE,
                    PHI,
                GROSSMM,
                  DELTA:  REAL;

BEGIN
   READ(KLEINEE,  GROSSMM);
   GROSSEE  :=  GROSSMM  { Startwert };
   SCHRITTNR  :=  0;
   REPEAT
      PHI  :=  GROSSMM  +  KLEINEE  *  SIN(GROSSEE);
      DELTA  :=  ABS(PHI  -  GROSSEE);
      SCHRITTNR  :=  SCHRITTNR  +  1;
      GROSSEE  :=  PHI { "E" fuer die naechste Iteration };
      WRITELN(SCHRITTNR,  GROSSEE:  20:12, DELTA:  20:  12);
   UNTIL  DELTA  <  EPSILON;
   WRITELN;
   WRITELN('  LOESUNG DER KEPLER-GLEICHUNG:');
   WRITELN('      GROSS E =',  GROSSEE:  15:  11)
END  {KEPLERGLEICHUNG}.
```

Ausgabe *(output)*

```
          1      0.595885107721      0.095885107721
          2      0.612248307096      0.016363199375
          3      0.614941766223      0.002693459127
          4      0.615382191501      0.000440425278
          5      0.615454128802      0.000071937301
          6      0.615465876625      0.000011747823
          7      0.615467795063      0.000001918438
          8      0.615468108346      0.000000313283
          9      0.615468159505      0.000000051159
         10      0.615468167859      0.000000008354
         11      0.615468169224      0.000000001364
         12      0.615468169446      0.000000000223
         13      0.615468169483      0.000000000036
         14      0.615468169489      0.000000000006

  LOESUNG DER KEPLER-GLEICHUNG:
     GROSS E =  0.61546816949
```

Bild 28. Programm KEPLERGLEICHUNG und deren Lösung für die Exzentrizität e=0,2 und M=0,5 rad. Nach 14 Iterationsschritten ist der Fehler (rechte Spalte) kleiner als 10^{-11}. Die mittlere Spalte enthält den Winkel E

Ein Vergleich eines Buchstabens mit einem anderen liefert einen von drei möglichen Werten: der bekannte Buchstabe ist vor oder nach dem gesuchten oder ist ihm gleich. Dafür bietet sich die Einführung eines Aufzählungstyps für die Variable POSITION mit den Namen VOR, GLEICH und NACH an (statt z. B. einer antiquierten Verschlüsselung mit 1, 2, 3). Das Programm BUCHSTABENRATEN (*Bild 30*) zeigt so außer den schon bekannten Elementen die Anwendung einer Auswahl-Anweisung. Der Anweisungsteil dieses Programms enthält eine Bedingungsschleife (*repeat ...until...;*), die so lange wiederholt wird, bis der Wert von POSITION gleich GLEICH ist (d. h. der logische Ausdruck POSITION = GLEICH erhält den Wert TRUE). Nach dem Ausdruck eines O. K. wird das Programm schließlich beendet. Innerhalb der Schleife geschieht folgendes: Es wird als erstes ein Buchstabe angefordert. Dann wird durch die bedingte Verzweigung (IF GEGEBEN = GESUCHT...) die Position des eingegebenen Buchstabens im Vergleich zum „gesuchten" (P) ermittelt. Die erste Ausgabeanweisung druckt den eingegebenen Buchstaben mit einem Text aus. Der mittlere Teil der Ausgabe ist von dem Wert der Variablen POSITION abhängig. Dazu wird die Auswahlanweisung herangezogen. Je nach ihrem Wert wird eine der drei „write"-Anweisungen ausgeführt. Daß innerhalb der Anführungszeichen für den anzugebenden Klartext (z. B. „GLEICH") wieder das Wort GLEICH vorkommt, möge den Leser nicht verwirren. Nach
CASE POSITION OF...

Programm

```
PROGRAM  ABZAEHLEN(INPUT,  OUTPUT);

VAR
                ZEICHEN:  CHAR;
                  VOKAL:  SET  OF  'A'..'Z';
                ZAEHLER:  0..  10000;

BEGIN
   VOKAL  :=  ['A',  'E',  'I',  'O',  'U'];
   ZAEHLER  :=  0;
   WHILE  NOT  EOF  DO
      BEGIN
         READ(ZEICHEN);
         IF  ZEICHEN  IN  VOKAL  THEN
            ZAEHLER  :=  ZAEHLER  +  1
      END  {WHILE};
   WRITE(ZAEHLER,  ' VOKALE GEFUNDEN')
END  {ABZAEHLEN}.
```

Eingabe *(input)*

```
PROGRAM ABZAEHLEN(INPUT,  OUTPUT);

VAR
                ZEICHEN:  CHAR;
                  VOKAL:  SET  OF  'A'..'Z';
                ZAEHLER:  0..  10000;

BEGIN
   VOKAL  :=  ['A',  'E',  'I',  'O',  'U'];
   ZAEHLER  :=  0;
   WHILE  NOT  EOF  DO
      BEGIN
         READ(ZEICHEN);
         IF  ZEICHEN  IN  VOKAL  THEN
            ZAEHLER  :=  ZAEHLER  +  1
      END  {WHILE};
   WRITE(ZAEHLER,  ' VOKALE GEFUNDEN')
END  {ABZAEHLEN}.
```

Ausgabe *(output)*

```
82 VOKALE GEFUNDEN
```

Bild 29. Programm ABZAEHLEN. Eingabe für das Programm ist die Programmliste selbst. Erklärung zur Menge VOKAL befindet sich im Text

bedeutet das erste Auftreten GLEICH: ...;, daß hier ein Wert des Aufzählungstyps angeführt ist und – falls POSITION = GLEICH gilt – die Anweisung nach dem Doppelpunkt ausgeführt wird. Das GLEICH innerhalb der Anführungszeichen zusammen mit den Leerzeichen ist dagegen ein Klartext, der direkt ausgegeben wird. Ein Text könnte an dieser Stelle auch 'IST DER GESUCHTE BUCHSTABE' lauten. Zu beachten ist im Programm BUCHSTABENRATEN die Wirkung von WRITE (...). Das erste WRITE (' " ', ...) gibt eine Zeichenkette aus, in der gleichen Zeile wird dann nach dem nächsten Prozeduraufruf innerhalb der Auswahlanweisung ein ' VOR ' o. a. angefügt. Diesem wird schließlich ' DEM ' angehängt, und es wird auf die nächste Zeile übergegangen (wegen WRITELN statt WRITE).

Die Auswahlstellen in der *case*-Anweisung sind also keine Marken, im Unterschied beispielsweise zu FORTRAN oder ALGOL. Sie können daher auch nicht mit einem *goto* angesprungen und dürfen natürlich nicht als Marken vereinbart werden. Sie sind Konstante, für die Werte der sogenannten geordneten Datentypen (mit den Standardtypen *integer, char* und *boolean)* in Frage kommen; bei *boolean* wären dies nur zwei Alternativen: FALSE und TRUE. Die Liste der Konstanten muß nicht alle Werte umfassen; gehören mehrere zusammen, so werden sie durch Kommas getrennt aneinandergereiht.

Falls nicht alle Werte des Ausdrucks nach dem Wortsymbol *case* (siehe Syntaxdiagramm Bild 20b) aufgelistet sind, kann es im Programmlauf passieren, daß die entsprechende Auswahlstelle bis zum abschließenden *end* nicht gefunden wurde. In diesem Fall wird die weitere Ausführung abgebrochen und eine Fehlermeldung zur *case*-Anweisung gegeben (etwa: Wert außerhalb des Bereiches). Diese harte Festlegung hat zum Teil heftige Kontroversen ausgelöst. Um dem Programmierer das Leben zu „erleichtern", wurden daher auch bei manchen PASCAL-Übersetzern Erweiterungen vorgenommen: Das, was unter „Sonstiges" fällt, wird nach *otherwise* bzw. *else* behandelt. In dem internationalen Entwurf zur Sprache PASCAL (Stand April 1980) sind keine Änderungen zu diesem Punkt

Programm

```
PROGRAM  BUCHSTABENRATEN(INPUT/,  OUTPUT);

VAR
                GESUCHT,
                GEGEBEN:  'A'  ..  'Z';
               POSITION:  (VOR,  GLEICH,  NACH);
                             { Aufzaehlungstyp }

BEGIN
   GESUCHT  :=  'P';
   WRITELN('  BITTE GEBEN SIE NACH JEDEM FRAGEZEICHEN');
   WRITELN('  EINEN BUCHSTABEN EIN!');
   REPEAT
      READLN;
      READ(GEGEBEN);
      IF: GEGEBEN  =  GESUCHT
      THEN
         POSITION  :=  GLEICH
      ELSE
         IF  GEGEBEN  <  GESUCHT
         THEN
            POSITION  :=  VOR
         ELSE
            POSITION  :=  NACH;
      WRITE('"',  GEGEBEN,  '" IST ');
      CASE  POSITION  OF
         VOR:
            WRITE('    VOR    ');
         GLEICH:
            WRITE('  GLEICH  ');
         NACH:
            WRITE('   NACH   ')
      END  {CASE};
      WRITELN(' DEM GESUCHTEN BUCHSTABEN');
   UNTIL: POSITION  =  GLEICH;
   WRITELN('  O.K.')
END  {BUCHSTABENRATEN}.
```

Interaktive Ein-/Ausgabe

```
          BITTE GEBEN SIE NACH JEDEM FRAGEZEICHEN
          EINEN BUCHSTABEN EIN!
?           "M" IST     VOR    DEM GESUCHTEN BUCHSTABEN
?           "T" IST    NACH    DEM GESUCHTEN BUCHSTABEN
?           "O" IST     VOR    DEM GESUCHTEN BUCHSTABEN
?           "Q" IST    NACH    DEM GESUCHTEN BUCHSTABEN
?           "P" IST   GLEICH   DEM GESUCHTEN BUCHSTABEN
          O.K.
```

Bild 30. Verbessertes Programm BUCHSTABENRATEN (3. Version) durch Einführen eines Aufzählungstyps. Nach wie vor ist der gesuchte Buchstabe einsehbar: Eine endgültige Version müßte den Buchstaben per Zufallsgenerator erzeugen. Dieses Programm wurde ausnahmsweise von einem Compiler für interaktives Arbeiten übersetzt. Der Leser möge die nichtstandardmäßigen Elemente (Schrägstrich nach INPUT und READLN vor READ(GEGEBEN)) entschuldigen. Aber nur dadurch war es auf dem vorhandenen Rechner möglich, den Buchstaben im „Dialog" zu raten. Das Fragezeichen wird nach dem READLN ausgegeben, dann tippt der Benutzer ein Zeichen ein (nicht gedruckt)

Eingabe *(input)*

```
20  3  1980
```

Programm

```
PROGRAM  KALENDER(INPUT,  OUTPUT);

CONST
          KORREKTUR  =  13;
          { Gregorianische Korrektur fuer
                die Jahre 1901 bis 2099 }

TYPE
               DATUM  =  RECORD
                                  TAG:  1..31;
                                MONAT:  1..12;
                                 JAHR:  1901..2099
                         END;

VAR
            SCHALTJAHR:  BOOLEAN;
                     D:  DATUM;
                    JD:  INTEGER;
             WOCHENTAG:  0..6  {Montag bis Sonntag};
             TAGESZAHL:  1..366;

BEGIN
   WITH  D  DO
      BEGIN
         READ(TAG,  MONAT,  JAHR);
         SCHALTJAHR  :=  (0  =  JAHR  MOD  4);
         IF  MONAT  >  2
         THEN
            TAGESZAHL := TRUNC(30.6 * (MONAT + 1)) -
                         63 + TAG + ORD(SCHALTJAHR)
         ELSE
            TAGESZAHL := TRUNC(30.6 * (MONAT + 13)) -
                         428 + TAG;
         JD := TRUNC((JAHR - 1) * 365.25) + TAGESZAHL
                  - KORREKTUR + 1721423;
      END  {WITH};
   WOCHENTAG  :=  JD  MOD  7;
   WRITELN('   JULIANISCHER TAG=    ',  JD);
   WRITELN('   NR. DES WOCHENTAGES=',  WOCHENTAG)
END  {KALENDER}.
```

Ausgabe *(output)*

```
JULIANISCHER TAG=       2444319
NR. DES WOCHENTAGES=          3
```

Bild 31. Programm KALENDER (1. Version). Es berechnet den Julianischen Tag und den Wochentag für den 20. 3. 1980, dem Erscheinungstag von Heft 6/80 der ELEKTRONIK. Da die Zählung des Wochentages mit 0 für Montag beginnt, ist 3 dann der Donnerstag

vorgesehen. Dem Programmierer sei empfohlen, lieber etwas mehr zu schreiben, als vorschnell auf Erweiterungen zurückzugreifen.

5.2.5 Inspektionsanweisung

Die Inspektionsanweisung („with"-Anweisung) dient zur Vereinfachung des Zugriffes auf eine Verbund-Variable. Wie bereits bei der Besprechung der Verbunde erläutert, geschieht das Ansprechen einer Verbund-Komponente durch das Anhängen eines Punktes an den Variablennamen, gefolgt von dem Bezeichner für die Komponente. Nach den Vereinbarungen

```
TYPE   DATUM = RECORD
                 TAG:   1..31;
                 MONAT: 1..12;
                 JAHR:  1901..2099
               END;
VAR    D : DATUM;
```

steht eine Datumsvariable D vom Typ DATUM zur Verfügung. Im Anweisungsteil könnte man mit D.TAG bzw. D.MONAT bzw. D.JAHR auf eine der Komponenten TAG usw. zugreifen. Eine Eingabeanweisung für ein Datum hätte dann folgende Gestalt:

```
READ (D.TAG, D.MONAT, D.JAHR).
```

Nacheinander verlangt das Programm ganze Zahlen für den Tag, den Monat und das Jahr (und prüft dabei die Einhaltung des jeweiligen Wertbereiches). Durch die Inspektionsanweisung, beginnend mit

```
WITH D DO...
```

wird innerhalb von *begin...end* auf eine Komponente der Variablen D so zugegriffen, daß nur noch der Komponentenbezeichner TAG usw. an Stelle von D.Tag usw. gesetzt wird.

Das nächste Programm bestimmt den Julianischen Tag. An Hand dieser von *Joseph Scaliger* (1540...1609) eingeführten Tageszählung lassen sich leicht chronologische Rechnungen durchführen (hauptsächlich in der Astronomie). Teilt man z. B. den Julianischen Tag durch sieben, so stellt der Rest den Wochentag (Rest: 0 = Mo, 1 = Di bis 6 = So) dar. In [18] ist dieses Thema behandelt und der im Programm KALENDER (*Bild 31*) verwendete Algorithmus angegeben. Um das Programm übersichtlicher zu gestalten, ist die Berechnung auf die Jahre 1901 bis 2099 beschränkt. Für diesen Zeitraum gilt die sog. Gregorianische Korrektur von K = 13; die anderen Werte von K für die Zeit von 4713 v. Chr. bis 3268 n. Chr. finden sich in dem erwähnten Zitat [18].

In KALENDER ist die logische Variable SCHALTJAHR vereinbart und erhält durch den Ausdruck

```
0 = JAHR MOD 4
```

einen Wert zugewiesen. In der linken Seite des Vergleiches wird der Rest der Jahreszahl auf 4 mit dem Operator *mod* berechnet. (Die runden Klammern um den Ausdruck dienen nur zur Hervorhebung, sind aber syntaktisch nicht notwendig.) Ist das Jahr ein Schaltjahr, so ist der Rest 0 und der Vergleich ergibt den Wert TRUE. Für die Berechnung der Tageszahl eines Jahres wird die Eigenschaft des Aufzählungstyps *boolean* genutzt: Die Ordnungszahl von FALSE ist 0 und die von TRUE 1. Schließlich enthält die Berechnung noch die Standardfunktion *trunc*, die aus dem reellwertigen Ausdruck

```
30.6 * (MONAT + 1)
```

auf die nächstkleinere ganze Zahl abrundet. Ein ganz korrektes Programm von KALENDER erfordert natürlich das Abprüfen des Datums auf richtige Tagesangaben (29. 2. 1980 ist zulässig, 31. 11. 1980 dagegen nicht). Durch den Unterbereichstyp 1...31 ist schon eine gewisse, aber keine vollständige Sicherheit gegen Fehlbedienung gewonnen. Im Kapitel 6 wird noch eine kleine anschließende Prozedur zur Ausgabe der Wochentage in Abhängigkeit von dem Wert WT angegeben.

6 Unterprogramme: Prozeduren und Funktionen

Die allgemein anerkannte Methode der strukturierten Programmierung zur Herstellung zuverlässiger Programme betont die große Wichtigkeit des richtigen Umsetzens eines Problems in Teilaufgaben. Prozeduren und Funktionen als Unterprogramme in PASCAL sollten daher immer dann eingeführt werden, wenn dadurch das Programm übersichtlicher, schneller und kürzer wird. Für eine Reihe von immer wieder auftretenden Aufgaben gibt es die sogenannten Standardprozeduren und -funktionen. Sie sind bereits im Abschnitt 3.4 (Tabelle 2) aufgeführt worden und dem Leser teilweise bereits bekannt (*read*, *write*, *exp* u. a.). Davon stellten die Unterprogramme zur Ein-/Ausgabe die Verbindung zwischen dem Rechner und dem Anwender her. Der größeren Effizienz wegen werden Teile dieser Standardprozeduren (bzw. -funktionen) in der Assemblersprache der jeweiligen Anlage geschrieben.

Der Hauptunterschied zwischen den beiden Unterprogrammformen liegt, wie schon erwähnt, in der Wirkung ihres Aufrufes: ein Prozeduraufruf führt den definierten Anweisungsteil aus. Der Prozedurname steht sozusagen für diesen Unterprogrammteil. Bei einem Funktionsaufruf dagegen wird dem Funktionsnamen ein Wert zugewiesen, der innerhalb eines Ausdruckes weiter verwendet wird. Dieser Unterschied kommt in den Vereinbarungsteilen zum Ausdruck. Da sonst keine prinzipiellen Differenzen auftreten, werden im folgenden beide Formen gemeinsam betrachtet. Den verschiedenen Parametern wird dabei besondere Aufmerksamkeit geschenkt.

6.1 Vereinbarung

Prozeduren und Funktionen werden in PASCAL nach den Vereinbarungsabschnitten für Marken, Konstanten, Typen und Variablen definiert. Dadurch erhalten die PASCAL-Programme ihr charakteristisches Aussehen: Nach dem meist längeren Vereinbarungs-

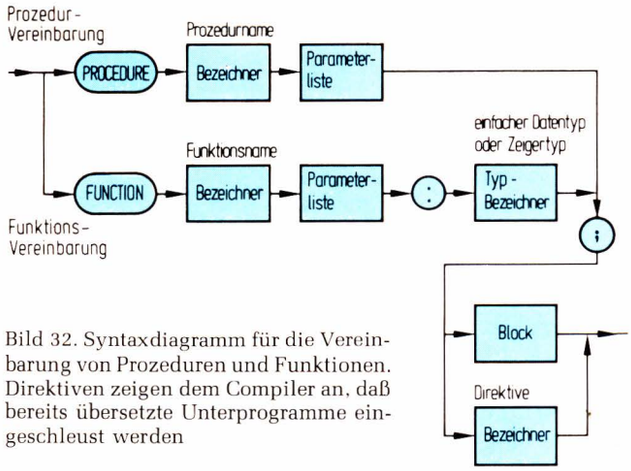


Bild 32. Syntaxdiagramm für die Vereinbarung von Prozeduren und Funktionen. Direktiven zeigen dem Compiler an, daß bereits übersetzte Unterprogramme eingeschleust werden

teil folgt am Schluß ein relativ kurzer Anweisungsteil des Hauptprogrammes (z. B. im Unterschied zu FORTRAN, dort sind die Unterprogramme zuletzt aufgeführt). Die Vereinbarungen für Prozeduren und Funktionen können – wie aus Bild 5 im Abschnitt 3.2 ersichtlich – in beliebiger Reihenfolge angeordnet sein; *Bild 32* und *33* zeigen die ausführlichen Syntaxdiagramme. Ungewöhnlich sieht im Vergleich zu ALGOL, FORTRAN u. a. die Parameterliste aus: Innerhalb der runden Klammern befinden sich nicht nur die Namen der formalen Parameter, sondern auch – nach dem Anhängen eines Doppelpunktes – die Angabe eines Typs. Diese für PASCAL und verwandte Sprachen typische Syntax bindet jeden Parameter sichtbar an einen Typ, der durch einen *Bezeichner* gegeben ist. Bei einem Standardtyp (*integer* usw.) ist dies problemlos. Alle anderen Typen (Aufzählungs-, Unterbereichs- und strukturierte Typen) verlangen einen eigenen Bezeichner, der in der Typ-Vereinbarung zuvor definiert worden ist.

X1:COMPLEX
ist eine zulässige Spezifikation eines formalen Parameters, falls COMPLEX vorher als (Verbund-)Typ vereinbart wurde.

X1: RECORD RE, IM: REAL END
ist dagegen falsch, weil nach dem Doppelpunkt eine Typ-Definition und nicht ein Typ-Bezeichner folgt (wie oben sollte X1 eine Verbundvariable sein). Dieser Umstand bringt es mit sich, daß der Anfänger in PASCAL vornehmlich Standardtypen einsetzt. Das ist natürlich einfacher, aber nicht immer dem Problem angemessen.

Natürlich gibt es Unterprogramme auch ohne Parameter. Dadurch wird die Vereinbarung und der Aufruf vereinfacht. Meist ist es jedoch sicherer, alle Größen für den Ein- und Ausgang eigens auszuweisen (beispielsweise zur besseren Testmöglichkeit des Programms).

Die im Prozedur- bzw. Funktionskopf eingeführten Parameter heißen definitionsgemäß *formale* Parameter. Beim Aufruf treten an ihre Stelle die *aktuellen* Parameter, die erstens vom gleichen Typ sein und zweitens an derselben Position innerhalb der Parameterliste stehen müssen.

Je nach Art der Parameterübergabe verwendet man
Werte-Parameter,
Variablen-Parameter,
Prozedur-Parameter und
Funktions-Parameter.

Diese Formen werden anschließend an Hand von Programmen erläutert.

Dem Prozedur- bzw. Funktionskopf folgt wahlweise ein Block oder eine Direktive. Ein Block besteht – wie schon im Bild 5 dargestellt – aus einem Vereinbarungs- und Anweisungsteil. Die vereinbarten Marken, Konstanten, Variablen, Typen und weiteren Prozeduren und Funktionen sind wie die formalen Parameter nur für den folgenden Anweisungsteil gültig. Man nennt sie daher *lokal*. Nach dem Abschluß eines Prozedur- oder Funktionsaufrufes ist ihre Lebensdauer zu Ende. Die vor einer Prozedur (bzw. Funktion) vereinbarten Größen sind im Gegensatz dazu global. Auf sie können auch die „inneren" Unterprogramme zugreifen. Mit entsprechenden Wertzuweisungen sind auch Änderungen der Werte von globalen Variablen möglich. Da solche Modifikationen oft schwer zu lokalisierende Fehler schaffen, sollte – falls möglich – auf globale Variable nur lesend zugegriffen werden (Stichwort: Seiteneffekt).

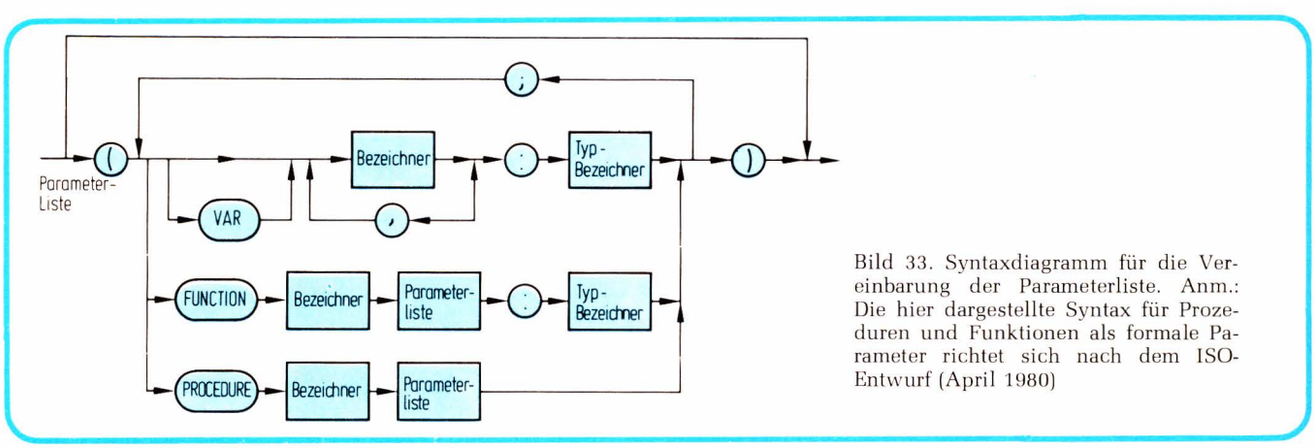


Bild 33. Syntaxdiagramm für die Vereinbarung der Parameterliste. Anm.: Die hier dargestellte Syntax für Prozeduren und Funktionen als formale Parameter richtet sich nach dem ISO-Entwurf (April 1980)

Anstelle eines Blocks kann eine Direktive treten. Dadurch wird dem Compiler mitgeteilt, daß das Unterprogramm bereits übersetzt vorliegt. Auf diese Weise sind externe Prozeduren und Funktionen in das Programm einschleusbar. Es gibt z. B. die anlagenspezifischen Direktiven

EXTERN
für externe PASCAL-Unterprogramme und

FORTRAN
für externe FORTRAN-Routinen.

Standardmäßig ist die sogenannte Vorwärtsdeklaration durch die Direktive FORWARD. Sie wird dann benutzt, wenn sich Prozeduren (bzw. Funktionen) wechselseitig aufeinander beziehen. Die Konstruktion

```
PROCEDURE A; FORWARD;
PROCEDURE B;
BEGIN ...; A; ... END;
PROCEDURE A;
BEGIN ...; B; ... END;
:
```

demonstriert für die parameterlosen Prozeduren A und B die Notwendigkeit der Vorwärtsvereinbarung von A, weil in B der Prozedurname A bekannt sein muß.

6.2 Werte-Parameter

Werte-Parameter („*value*"-Parameter) als einfachste Form übergeben an die Prozedur (Funktion) Werte, ohne daß er dann geändert werden darf (und kann!). Die Standardprozedur

WRITE (JD)

bewirkt bekannterweise die Ausgabe der aktuellen Zahl der „integer"-Variablen JD. JD hat nach der Ausgabe denselben Wert wie vorher und kann weiter im Programm verarbeitet werden.

Das nächste Beispiel (*Bild 34*) zeigt die erste selbstdefinierte Funktion. Für die Berechnung der Entropie einer Nachrichtenquelle nach Shannon

$$H = -\sum_{i=1}^{n} p_i \, \mathrm{ld}\, p_i$$

wird der duale Logarithmus benötigt. Da aber nur der natürliche Logarithmus (LN) standardmäßig vorhanden ist, wird die Funktion LOG2 (= lnx/ln2) eingeführt. Nur wegen der Beschränkung auf Großbuchstaben wurde sie nicht mit LD (für ld) bezeichnet.

Innerhalb des kurzen Anweisungsteiles der Funktion LOG2 sind X und LNOFTWO lokale Variable, bzw. Konstanten. Auf die globalen Variablen PROBABILITY, SUMOFPROBABILITIES, H, NUMBER, INPUT und OUTPUT wird überhaupt nicht zugegriffen. Anstelle des formalen Parameters X vom Typ „real" wird beim Funktionsaufruf LOG2 (PROBABILITY) als aktueller Parameter PROBABILITY eingesetzt. Anmerkung: Den Zahlenwert der Konstanten LNOFTWO kann man sich durch das kurze Programm

```
PROGRAM T(OUTPUT);
BEGIN WRITE (LN(2.0)) END.
```

schnell beschaffen, falls kein Taschenrechner greifbar und/oder eine höhere Genauigkeit erwünscht ist.

Ein gewisser Schutz gegen eine fehlerhafte Eingabe – zuviele oder zuwenige Ereignisse p_i – ist durch die Abfrage

$$\left| 1 - \sum_{i=1}^{n} p_i \right| < 10^{-6}$$

in der bedingten Verzweigung erreicht. Dem Programm fehlt jedoch ein Test auf negative Werte für PROBABILITY, da dann LN(X) nicht mehr (und alles weitere) berechnet werden kann.

Die Vereinbarung einer Funktion in Bild 34 demonstriert hier natürlich nicht den platz- und zeitsparenden Einsatz eines Unterprogrammes. Es betont mehr die Übersichtlichkeit und zeigt das Vorgehen, wie

Eingabe *(input)*

```
0.140625
0.09375
0.0625
0.140625
0.09375
0.140625
0.09375
0.140625
0.09375
```

Programm

```
PROGRAM  ENTROPY(INPUT,  OUTPUT);

VAR
                H,       {average information content}
       PROBABILITY,
SUMOFPROBABILITIES:  REAL;
           NUMBER:  INTEGER;


FUNCTION  LOG2(X:  REAL):  REAL;

   CONST
           LNOFTWO  =  6.9314718055994E-1;
                          { =  ln(2)}

   BEGIN
      LOG2  :=  LN(X)  /  LNOFTWO
   END  {LOG2};


BEGIN  {ENTROPY}
   H  :=  0.0;
   SUMOFPROBABILITIES  :=  0.0;
   NUMBER  :=  0;
   WHILE  NOT  EOF  DO
      BEGIN
         READLN(PROBABILITY);
         SUMOFPROBABILITIES  :=
                    SUMOFPROBABILITIES  +  PROBABILITY;
         H  :=  H  -  LOG2(PROBABILITY)  *  PROBABILITY;
         NUMBER  :=  NUMBER  +  1
      END  {WHILE};
   WRITELN('  NUMBER OF EVENTS',  NUMBER:  6);
   IF  ABS(SUMOFPROBABILITIES  -  1.0)  <  1.0E-6
   THEN
      WRITELN('  ENTROPY      ',  H:  14:  4,  ' BIT')
   ELSE
      WRITELN('  ERROR:  SUM OF PROBABILITIES <> 1.0',
         SUMOFPROBABILITIES:  10:  6)
END  {ENTROPY}.
```

Ausgabe *(output)*

```
NUMBER OF EVENTS      9
ENTROPY               3.1226 BIT
```

Bild 34. Programm ENTROPY für die Berechnung des mittleren Informationsgehaltes mit der selbstdefinierten Funktion LOG2.

man sich nichtstandardmäßige Funktionen verschafft. Auf ähnliche Weise kann man die EXOR-Verknüpfungen im Programm PSEUDOZUFALLSFOLGEN (Bild 26) im Abschnitt 5.2.3.1 durch ein kleines Unterprogramm ersetzen. Die Vereinbarung könnte so aussehen:

```
FUNCTION EXOR (V1,V2:BOOLEAN): BOOLEAN;
BEGIN
    EXOR := (V1 AND NOT V2) OR
        (NOT V1 AND V2)
END {EXOR};
```

Auch hier besteht der Anweisungsteil nur aus einer Wertzuweisung an den Funktionsbezeichner EXOR (entweder FALSE oder TRUE). Ein Aufruf in dem geänderten Programm PSEUDOZUFALLSFOLGE sieht dann so aus:

```
SR[0] := EXOR (SR[N],SR[ABGRIFF]);
```

Als aktuelle Werteparameter für die formalen Parameter V1 und V2 sind hier indizierte Feldelemente von SR (als ARRAY[0...MAXSTUFEN] OF BOOLEAN vereinbart) eingesetzt. Entscheidend dabei ist, daß die Datentypen der Parameter identisch sind (hier: BOOLEAN). In dem gleichen Programm bietet es sich an, die Berechnung von Potenzen zur Basis 2 als Funktion zu formulieren. Der Leser möge dies vielleicht selbst versuchen.

Das nächste Programm KALENDER (*Bild 35*) ermöglicht dem Leser einen Vergleich zu der ersten Version ohne Prozedur in Bild 31. Durch die Prozedur SCHREIBWOCHENTAG wird die Ausgabe des Programms verständlicher. Die Parameterliste enthält neben einem Werte-Parameter zusätzlich einen Variablen-Parameter (s. u.). Für den Werte-Parameter WOTAG wurde ein angepaßter Unterbereichstyp eingeführt:

```
NULLSECHS = 0..6;
```

obwohl die Modulo-Operation (JD MOD 7) nur Werte zwischen 0 und 6 liefert. Bei künftigen Erweiterungen bleibt diese Prozedur sicher, weil die Auswahlweisung nur 0...6 erhält. Da in diesem Programm die Variable WOCHENTAG nur einmal einen numerischen Wert erhält, wurde auf die Definition eines Aufzählungstyps WOCHE = (MON,...) verzichtet. Bei einem Vorhandensein von WOCHE wäre in der Prozedur SCHREIBWOCHENTAG nur wenig zu ändern: statt NULLSECHS steht dann WOCHE; statt 0: ... dann MON:... usw. Neu zu programmieren wäre dagegen eine Funktion, die aus den Werten 0...6 die Werte MON...SON bestimmt. Ein Vorschlag:

```
FUNCTION W (WOTAG: NULLSECHS) : WOCHE;
BEGIN CASE WOTAG OF
    0: W:= MON;
    1: W:= DIE; ...END {CASE}
END {W};
```

Ebenso wie hier in KALENDER kann eine Prozedur zur Ausgabe der Variablen POSITION im Programm BUCHSTABENRATEN (Bild 30) eingefügt werden.

6.3 *Variablen-Parameter*

Das Wortsymbol *var* leitet in der Parameterliste einer Prozedur die Definition eines *Variablen-Parameters* ein. Diese Form kommt dann zum Tragen, wenn die Prozedur (bzw. die Funktion) den Wert des Parameters selbst verändert und an den aufrufenden Programmteil zurückgibt. Als aktueller Parameter kann natürlich nur eine Variable (keine Konstante) beim Aufruf eingesetzt sein. Die Unterscheidung zwischen Wert- und Variablen-Parametern bringt einen großen Gewinn an Sicherheit, weil damit der Programmierer möglichen Wertänderungen von Variablen an falscher Stelle vorbeugt. Bei Dateien als Parametertyp muß grundsätzlich ein Variablen-Parameter eingesetzt werden (Begründung: die aktuelle Position

Eingabe *(input)*

```
3  4  1980
```

Programm

```
PROGRAM  KALENDER(INPUT,  OUTPUT);

CONST
          KORREKTUR  =  13;
          { Gregorianische Korrektur fuer
               die Jahre 1901 bis 2099 }

TYPE
               DATUM  =  RECORD
                                          TAG:  1..  31;
                                        MONAT:  1..  12;
                                         JAHR:  1901..  2099
                         END;
          NULLSECHS  =  0..  6  {Montag bis Sonntag};

VAR
           SCHALTJAHR:  BOOLEAN;
                    D:  DATUM;
                   JD:  INTEGER;
          WOCHENTAG:  NULLSECHS;
          TAGESZAHL:  1..  366;

PROCEDURE  SCHREIBWOCHENTAG(  VAR  AUS:  TEXT;
                                   WOTAG:  NULLSECHS);

   BEGIN
      CASE  WOTAG  OF
         0:  WRITE(AUS,  'MONTAG    ');
         1:  WRITE(AUS,  'DIENSTAG  ');
         2:  WRITE(AUS,  'MITTWOCH  ');
         3:  WRITE(AUS,  'DONNERSTAG');
         4:  WRITE(AUS,  'FREITAG   ');
         5:  WRITE(AUS,  'SAMSTAG   ');
         6:  WRITE(AUS,  'SONNTAG')
      END  {CASE}
   END  {SCHREIBWOCHENTAG};

BEGIN
   WITH  D  DO
      BEGIN
         READ(TAG,  MONAT,  JAHR);
         SCHALTJAHR  :=  0  =  JAHR  MOD  4;
         IF  MONAT  >  2
         THEN
            TAGESZAHL  :=  TRUNC(30.6  *  (MONAT  +  1))  -
               63  +  TAG  +  ORD(SCHALTJAHR)
         ELSE
            TAGESZAHL  :=  TRUNC(30.6  *  (MONAT  +  13))  -
               428  +  TAG;
         JD  :=  TRUNC((JAHR  -  1)  *  365.25)  +  TAGESZAHL
            -  KORREKTUR  +  1721423;
         WRITE(TAG:  5,  '.',  MONAT:  3,  '.',  JAHR:  5);
      END  {WITH};
   WOCHENTAG  :=  JD  MOD  7;
   WRITE('   FAELLT AUF EINEN ');
   SCHREIBWOCHENTAG(OUTPUT,  WOCHENTAG);
   WRITELN;
   WRITELN('   JULIANISCHER TAG=',  JD)
END  {KALENDER}.
```

Ausgabe *(output)*

```
     3.  4. 1980   FAELLT AUF EINEN DONNERSTAG
    JULIANISCHER TAG=   2444333
```

Bild 35. Programm KALENDER (2. Version). Vergleiche dazu die Version in Bild 31 (Abschnitt 5.2.5) ohne Prozedur

einer Datei wird sowohl durch Lesen als auch Schreiben verändert).

Bevor das nächste Programm betrachtet wird, sei darauf hingewiesen, daß in der Eingabeanweisung die typische Verwendung eines Variablen-Parameters vorliegt. Ein Prozeduraufruf bewirkt entsprechend der Dateneingabe die Zuweisung eines Wertes an den Parameter. Der frühere Wert des Parameters ist dann unwiderruflich gelöscht.

Das Programm KOMPLEXE ARITHMETIK (*Bild 36*) greift die komplexe Rechnung mit dem Verbund COMPLEX auf. In diesem Beispiel ist eine Prozedur für die verschiedenen Operationen eingeführt. Eine andere Lösung bestände darin, für jede der Operationen eine eigene Prozedur zu schreiben. Zur Vereinfachung und Platzersparnis der Eingabe des Operationssymbol wurde hier der Standardtyp „char" für die Variable OPERATOR herangezogen. Das Programm enthält vor der Auswahlanweisung einen Test, ob der übergebene Wert von OPERATOR einer der vier programmierten Alternativen ist. Ist das Zeichen außerhalb der Menge

|'+', '–', '*', '/'|,

so erfolgt eine Fehlermeldung und die Komponenten der Verbundvariablen RESULTAT erhalten den Wert „undefiniert" (das ist exakter als die in anderen Sprachen übliche Vorbesetzung mit Nullen). Auf die Behandlung einer Division durch Null (NENNER = 0) wurde aus Platzgründen verzichtet. Im Regelfall – also OPERATOR ist in der Menge enthalten – erfolgt die gewünschte arithmetische Berechnung und führt zu einer Wertzuweisung an den Variablen-Parameter RESULTAT. Beim Verlassen der Prozedur werden die Werte (Real- und Imaginärteil) an den aktuellen Parameter ERGEBNIS übergeben. Im Gegensatz dazu sind die Werteparameter ZAHL1, ZAHL2 und OPSYMBOL unverändert geblieben.

Das eingegebene Sternchen vor den beiden Zahlen 1,1+j0,4 und 0,5+j2,0 bedeutet durch die Definition der Prozedur CPLX, daß die komplexen Zahlen miteinander multipliziert werden. Man könnte dafür auch andere Zuordnungen – u. U. entgegen der gewohnten Schreibweise – wählen. Im Unterschied dazu ist das „★" in den Ausdrücken des Programms ein Grundsymbol für die Multiplikation *reeller* Größen.

Die Prozedur CPLX in Bild 36 greift bis auf die Text-Datei *output* nur auf lokale Variable zu, so daß eine klare Verbindung zwischen Haupt- und Unterprogramm über die Parameter besteht. Bei der Übernahme von CPLX in andere Programme muß man beachten, daß die Typvereinbarung für COMPLEX mitgenommen werden muß.

Das Programmbeispiel macht gleichzeitig auch den Unterschied zu den Funktionen deutlich: Der Prozeduraufruf ist eine eigenständige Anweisung. Dem Prozedurbezeichner wird nirgends etwas zugewiesen; dagegen aber einem Funktionsnamen. CPLX als Funktion zu definieren ist nicht möglich, da der Resultattyp einer Funktion nur ein einfacher Datentyp oder ein Zeigertyp sein darf. COMPLEX ist ein strukturierter Typ (Verbund), deshalb ist in PASCAL

FUNCTION CPLX (...) : COMPLEX;

unzulässig. Diese wünschenswerte Eigenschaft wird manchmal – wie hier – vermißt. Die Beschränkung wurde von *N. Wirth* zugunsten der Einfachheit der Sprache vorgenommen und wird allgemein akzeptiert. Aus dem gleichen Grunde gibt es keine Namens-Parameter (wie z. B. in ALGOL). Wo diese Programmiertechnik gebraucht wird, setzte man Prozedur- und Funktionsparameter ein.

Eingabe *(input)*

```
*
1.1  0.4  0.5  2.0
```

Programm

```
PROGRAM  KOMPLEXEARITHMETIK(INPUT,  OUTPUT);

TYPE
            COMPLEX  =  RECORD
                                RE,
                                IM:  REAL
                        END;

VAR
                ZAHL1,
                ZAHL2,
            ERGEBNIS:  COMPLEX;
            OPSYMBOL:  CHAR;

PROCEDURE  CPLX(     OPERAND1,
                     OPERAND2:  COMPLEX;
                     OPERATOR:  CHAR;
              VAR  RESULTAT:  COMPLEX);

   VAR
                  NENNER:  REAL;

   BEGIN
      IF  OPERATOR  IN  ['+',  '-',  '*',  '/']
      THEN
    CASE  OPERATOR  OF
     '+': BEGIN
            RESULTAT.RE  :=  OPERAND1.RE  +  OPERAND2.RE;
            RESULTAT.IM  :=  OPERAND1.IM  +  OPERAND2.IM
         END  {'+'};
     '-': BEGIN
            RESULTAT.RE  :=  OPERAND1.RE  -  OPERAND2.RE;
            RESULTAT.IM  :=  OPERAND1.IM  -  OPERAND2.IM
         END  {'-'};
     '*': BEGIN
            RESULTAT.RE  :=  OPERAND1.RE * OPERAND2.RE  -
               OPERAND1.IM  *  OPERAND2.IM;
            RESULTAT.IM  :=  OPERAND1.RE * OPERAND2.IM  +
               OPERAND1.IM  *  OPERAND2.RE
         END  {'*'};
     '/': BEGIN
            NENNER := SQR(OPERAND2.RE) + SQR(OPERAND2.IM);
            RESULTAT.RE  :=  (OPERAND1.RE * OPERAND2.RE  +
               OPERAND1.IM  *  OPERAND2.IM)  /  NENNER;
            RESULTAT.IM  :=  (OPERAND2.RE * OPERAND1.IM  -
               OPERAND1.RE  *  OPERAND2.IM)  /  NENNER
            END  {'/'}
      END  {CASE}
   ELSE
            WRITELN(OUTPUT,  '  OPERATOR UNZULAESSIG!')
      END  {CPLX};

BEGIN  {KOMPLEXEARITHMETIK}
   {der Operator wird in der ersten Eingabezeile eingegeben}
   READLN(OPSYMBOL);
   READ(ZAHL1.RE,  ZAHL1.IM,  ZAHL2.RE,  ZAHL2.IM);
   CPLX(ZAHL1,  ZAHL2,  OPSYMBOL,  ERGEBNIS);
   WRITELN('  REALTEIL= ',  ERGEBNIS.RE:  12,
           '  IMAGINAERTEIL= ',  ERGEBNIS.IM:  12)
END  {KOMPLEXEARITHMETIK}.
```

Ausgabe *(output)*

```
REALTEIL=  -2.500E-001  IMAGINAERTEIL=   2.400E+000
```

Bild 36. Programm KOMPLEXE ARITHMETIK als Beispiel für einen Variablen-Parameter (VAR RESULTAT:COMPLEX). Der Kürze wegen wurde auf eine einfachere Ein- und Ausgabe verzichtet

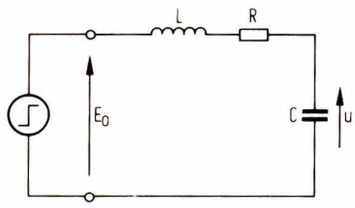


Bild 37. Ein LRC-Reihenresonanzkreis und seine Differentialgleichung; sprungförmig im Zeitpunkt Null mit E_o erregt

6.4 *Prozedur- und Funktionsparameter*

Eine Reihe von Problemstellungen verlangen, daß Prozeduren und Funktionen selbst als Parameter einer Prozedur (bzw. Funktion) auftreten müssen. Ein typisches Beispiel dafür ist die Integration: Ein universelles Programm dazu entsteht nur dann, wenn unterschiedliche Funktionen integriert werden können. Ein anderes Beispiel ist die Lösung einer Differentialgleichung: Im nächsten Programm wird das Einschwingverhalten eines verlustbehafteten Schwingkreises (*Bild 37*) ermittelt. Die numerische Berechnung der Sprungantwort bringt natürlich für konstante Koeffizienten T_1 und T_2 keinen Vorteil, da die Lösungsgleichung seit langem bekannt ist. Sie ist jedoch dann von großem Nutzen, wenn nichtlineare oder instationäre Elemente vorliegen. Eine entsprechende Modifikation des Programms ist dann leicht möglich.

Das Programm SCHWINGKREIS in *Bild 38* demonstriert zwei neue Möglichkeiten der Anwendung von Prozeduren (für Funktionen gilt entsprechendes). Zum ersten ist in einer der Prozeduren, nämlich in DIFSY2, ein Prozedurparameter (FKT) enthalten. Zum zweiten ist diese Prozedur DIFSY2 eine Bibliotheksprozedur. Letzteres aus zwei Gründen: die Wiedergabe ihres Anweisungsteils verbietet sich hier aus Platzgründen (Länge: 200 Zeilen); und als Bibliotheks-Prozedur wird sie natürlich auch von anderen Benutzern des Rechners zur Lösung eines Differentialgleichungssystems verwendet. Statt des umfangreichen Prozedurrumpfes steht also die Direktive EXTERN. Nach der Übersetzung muß die bereits vorübersetzte PASCAL-Prozedur dem Hauptprogramm zugeladen werden. Erst dann ist SCHWINGKREIS startfähig.

Der Algorithmus für DIFSY2 beruht auf dem Extrapolationsverfahren [19] und führt einen Integrationsschritt zur Lösung N gewöhnlicher Differentialgleichungen 1. Ordnung

$$\frac{dy_1}{dx} = f_1\,(x,y_1,y_2,\ldots,y_N);$$

$$\frac{dy_2}{dx} = f_2(x,y_1,y_2,\ldots y_N)\ \ldots \qquad \frac{dy_N}{dx} = f_N(x,y_1,y_2,\ldots y_N)$$

aus ($N \leqq 33$). Die Berechnung eines Schrittes von y_i (x) zu y_i ($x + h_o$) wird dann beendet, wenn die vorgegebene Genauigkeitsschranke ε unterschritten ist (oder es wird die Schrittweite h_o halbiert). Das hier im Prozeduraufruf gewählte $\varepsilon = 10^{-10}$ soll nur die Güte des Verfahrens andeuten; für die grafische Ausgabe ist dies offensichtlich übertrieben. Das Feld Y enthält die y_i (x) und ist ein Variablen-Parameter. S ist ein

Eingabe *(input)*

```
1.0  1.0  0.3
```

Programm

```
PROGRAM  SCHWINGKREIS(INPUT,  OUTPUT);

CONST            NMAX  =  33;

TYPE            INDEX  =  1..  NMAX;
               VEKTOR  =  ARRAY  [INDEX]  OF  REAL;
VAR
                   ZEIT,
          INDUKTIVITAET,
             KAPAZITAET,
             WIDERSTAND,
                  ENULL,
                     T1,
                     T2,
                  DELTA:  REAL;
                      N:  INDEX;
                    MUE:  INTEGER;
                    SPG,
                      S:  VEKTOR;
PROCEDURE  DGL(                T:  REAL;
                               U:  VEKTOR;
                   VAR  DUDT:  VEKTOR);
            {lesender Zugriff auf die globalen Variablen
             ENULL, T1 und T2}
   BEGIN
      DUDT[1]  :=  U[2];
      DUDT[2]  :=  (- U[2] * T1  -  U[1]  +  ENULL) / T2
   END  {DGL};
PROCEDURE  DIFSY2(             N:  INDEX;
                     EPSILON:  REAL;
                      VAR  X,
                          HO:  REAL;
                      VAR  Y,
                           S:  VEKTOR;
              PROCEDURE  FKT(          XX:  REAL;
                                       YY:  VEKTOR;
                            VAR  DYYDXX:  VEKTOR));
   EXTERN;   {Direktive fuer Bibliotheks-Prozedur}
BEGIN  {SCHWINGKREIS}
   READ(INDUKTIVITAET,  KAPAZITAET,  WIDERSTAND);
   T1  :=  WIDERSTAND  *  KAPAZITAET;
   T2  :=  INDUKTIVITAET  *  KAPAZITAET;
   WRITELN('  T1=  ',  T1:  11);
   WRITELN('  T2=  ',  T2:  11);
   WRITELN;
   ENULL  :=  1.0;
   SPG[1]  :=  0.0;    {Anfangswerte des Vektors SPG}
   SPG[2]  :=  0.0;
   N  :=  2;
   S[1]  :=  0.0;      {Anfangswerte des Hilfsvektors}
   S[2]  :=  0.0;
   WRITELN('*':  ROUND(SPG[1]  *  40));
   FOR  MUE:=0  TO  40  DO
      BEGIN
         DELTA  :=  0.4;  {starre Schrittweite}
         ZEIT  :=  DELTA  *  MUE;
         DIFSY2(N,  1E-10,  ZEIT,  DELTA,  SPG,  S,  DGL);
         WRITELN('*':  ROUND(SPG[1]  *  40))
      END  {FOR}
END  {SCHWINGKREIS}.
```

Ausgabe *(output)*

```
T1=  3.00E-001
T2=  1.00E+000
```

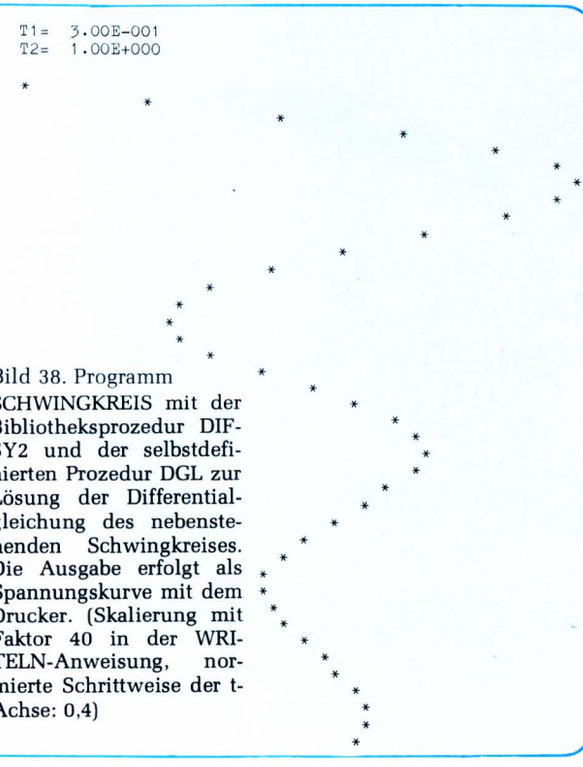


Bild 38. Programm SCHWINGKREIS mit der Bibliotheksprozedur DIFSY2 und der selbstdefinierten Prozedur DGL zur Lösung der Differentialgleichung des nebenstehenden Schwingkreises. Die Ausgabe erfolgt als Spannungskurve mit dem Drucker. (Skalierung mit Faktor 40 in der WRITELN-Anweisung, normierte Schrittweise der t-Achse: 0,4)

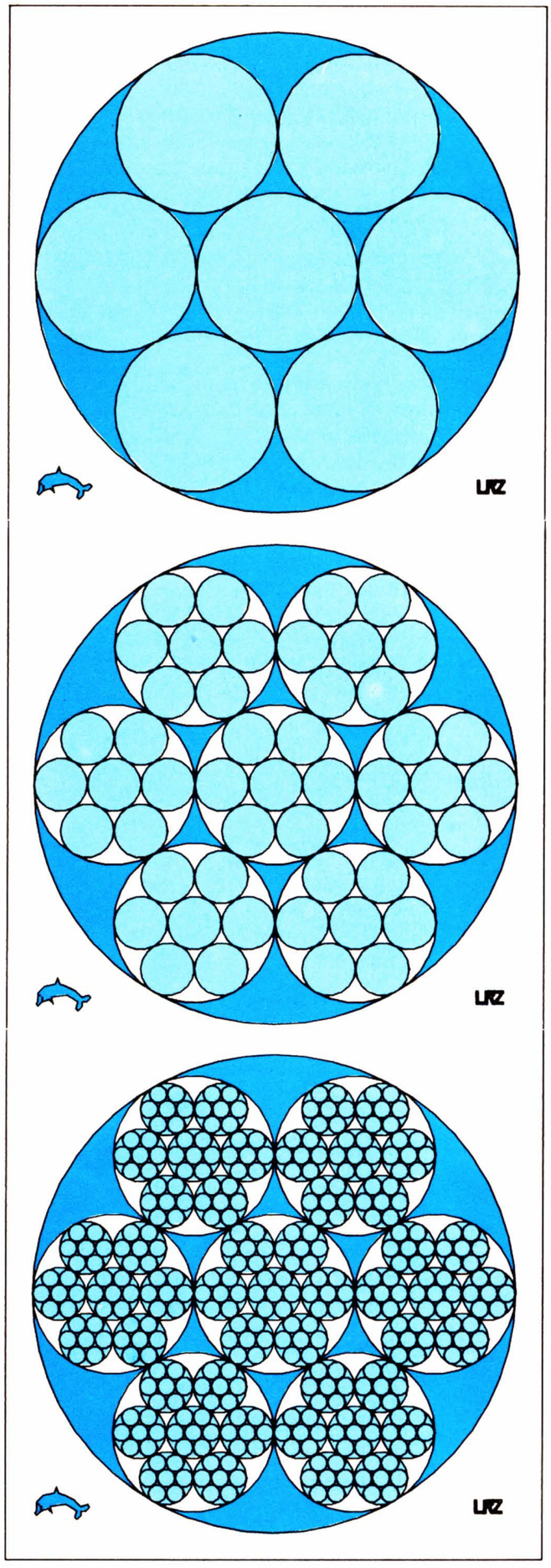


Bild 39. Drei Grafik-Zeichnungen von der Figur des „Siebenkreises" mit 1, 2 und 3 Schichten

Hilfsfeld für die Konvergenzbestimmung und erhält beim Aufruf die gleichen Anfangswerte wie Y.

Der letzte Parameter in DIFSY2 ist selbst eine Prozedur. Diese Prozedur FKT berechnet die Ableitungen dy_i/dx für das augenblickliche x und y_i. Für das gegebene Problem muß die Differentialgleichung 2. Ordnung nach bekannter Weise in zwei Differentialgleichungen 1. Ordnung umgewandelt werden.

Aus

$$T_2 \frac{d^2u}{dt^2} + T_1 \frac{du}{dt} + u = E_o$$

erhält man nach dem Einsetzen von

$$u_1(t) = u(t) \text{ und}$$

$$u_2(t) = \frac{du(t)}{dt}$$

folgendes Differentialgleichungssystem (N=2):

$$\frac{du_1}{dt} = u_2;$$

$$\frac{du_2}{dt} = \frac{1}{T_2} (-T_1 \cdot u_2 - u_1 + E_o);$$

Diese Gleichungen sind also speziell für den Schwingkreis und werden in der Prozedur DGL berechnet. Damit hat das Programm SCHWINGKREIS zwei Prozeduren. Bei jedem Aufruf von DIFSY2 wird für den formalen Parameter FKT (eine Prozedur) der aktuelle Prozedur-Parameter DGL eingesetzt.

Das Hauptprogramm enthält nach dem Setzen der Anfangswerte im wesentlichen eine Laufanweisung zur Ausführung von insgesamt 41 Integrationsschritten. Die Ausgabe der Lösungskurve für die Spannung $u_1(t)$ (=u(t); SPG |1| im Programmtext) ist grafisch und geschieht durch die Steuerung der Länge der Zeichenkette. '*':10 z. B. heißt, daß die Gesamtlänge 10 ist, davon 9 Leerzeichen und rechtsbündig das Zeichen „*".

Anmerkung: Die Vereinbarung des Prozedurparameters FKT in DIFSY2 geschieht hier nach dem neuesten Entwurf zur Standardisierung von PASCAL (ISO-Entwurf vom April 1980). Im Unterschied zum früheren Standard wird die Parameterliste der Prozedur FKT angeführt. Dies bringt eine erhebliche Erleichterung beim Anschluß von Bibliotheksprozeduren.

Der angesprochene Entwurf enthält als zweite wesentliche Änderung von PASCAL die Einführung von dynamischen Feldgrenzen (innerhalb einer Parameterliste). Auch dies erfolgte unter dem Gesichtspunkt, PASCAL-Programme praktischen Erfordernissen anzupassen (in der Lehre spielt dieser Punkt eine untergeordnete Rolle). Außer diesen beiden Modifikationen wird der internationale Standard keine nennenswerten Änderungen gegenüber bisher – und dem hier vorgestellten PASCAL – enthalten. Daß Computerhersteller sowohl Erweiterungen als auch Einschränkungen für ihr PASCAL anbieten, ist natürlich selbstverständlich. Diese sollten für den Anwender gut dokumentiert werden.

6.5 Standardprozeduren und -funktionen

Dieser Abschnitt kann hier kurz gehalten werden, weil der Leser einige der Unterprogramme in den Programmbeispielen kennengelernt hat. Die elementaren Datei-Operationen *get(.), put(.), reset(.), rewrite(.), eof(.)* sind bereits in diesem Sonderheft beschrieben worden [20]. Sie werden wegen der Puffervariablen – die jeder Datei beigeordnet ist – im Abschnitt 7 bei den Zeigern praktisch vorgestellt. Kennzeichnend für die Standardprozeduren und -funktionen ist einfach, daß sie in dem Programm vor dem Benutzerprogramm – d. h. im Compiler – vereinbart sind. Der Programmierer muß daher die genaue Spezifikation in den Handbüchern des Compilerherstellers nachsehen (siehe auch Tabelle 2 im 4. Abschnitt dieser Arbeit).

6.6 Rekursion

Zum Abschluß des 6. Kapitels soll von den diffizileren Programmiertechniken die Rekursion betrachtet werden. Als rekursiv bezeichnet man die Objekte, die sich selbst teilweise enthalten oder durch sich selbst definiert sind [21] (dieses anspruchsvolle Buch kann jedem fortgeschrittenen Programmierer empfohlen werden). PASCAL ermöglicht sowohl rekursive Kontrollstrukturen mit Prozeduren und Funktionen, als auch rekursive Datenstrukturen mit Zeigern (Abschnitt 7). Obwohl jede Rekursion auf eine Iteration zurückgeführt werden kann (und dann auch sollte, z. B. bei der Berechnung der Fakultät), ist sie für manche Problemstellungen nützlich.

In *Bild 39* ist das Objekt ein „Siebenkreis" [22]. Zu seiner Zeichnung wurde auf die Grafik-Bibliothek zurückgegriffen; die Prozeduren daraus sind im Programm *Bild 40* kurz erläutert.

Die benutzerdefinierte Prozedur KREIS7 ruft sich selbst auf, um in dem aktuellen Kreis (x,y) sieben weitere Kreise mit dem Radius r/3 zu erzeugen. Die Rekursion ist bei der Schicht 1 beendet. Schicht 0 bedeutet: ein großer Kreis ohne innere und äußere Siebenkreise; Schicht 1: in der vorhergehenden Schicht 0 sind sieben Kreise enthalten usw. Mögliche Schichten –1,–2... – die größer werdenden Kreise liegen an Schicht 0 – sind hier nicht programmiert. Nach unten werden die Schichten durch die Auflösung des Zeichengerätes begrenzt, nach oben durch die Größe des Zeichenbogens; die Rekursion selbst könnte theoretisch unendlich viele Schichten liefern.

Dieses Programm ist ein Beispiel für eine direkte Rekursion. Indirekte Rekursion erfordern die schon erwähnte Vorwärtsdeklaration im Abschnitt 6.1 (ohne weitere Besprechung).

Rückblickend auf diesen Abschnitt 6 kann nun gesagt werden, daß man ein PASCAL-Programm als eine Prozedur-Vereinbarung auffassen kann. Mit dem Unterschied, daß diese „Prozedur" durch das Wortsymbol *program* eingeleitet wird und die Parameterliste stark vereinfacht ist. Sie enthält nur Dateinamen ohne Typspezifikation.

Programm

```
PROGRAM  SIEBENKREISE  (INPUT,  OUTPUT,  PLOT);

CONST       LRZSYMBOL  =  139  {Symbolnr. fuer Leibniz-Rechenzentrum};

VAR              PLOT  :  TEXT  {Zeichendatei};
               SCHICHT  :  INTEGER;

PROCEDURE  SHEET   ( VAR PL : TEXT ; MST,  WE,  HO :  REAL); EXTERN;
   {Reservieren eines Zeichenbogens und Definition des Fensters}
PROCEDURE  SIZE    ( VAR PT : TEXT ; HOEHE: REAL);          EXTERN;
   {Festlegung der Groesse eines Schriftzeichens}
PROCEDURE  MARK    ( VAR PF : TEXT ; NR : INTEGER );        EXTERN;
   {Zeichnen eines Schriftzeichens}
PROCEDURE  DELPHIN ( VAR PF : TEXT ; MST,  X , Y :  REAL);  EXTERN;
   {Zeichnen eines Delphins}
PROCEDURE  FRAME   ( VAR PF : TEXT );                       EXTERN;
   {Einrahmen des Fensters}
PROCEDURE  CIRCLE  ( VAR PF : TEXT ; R, PHI1, PHI2:  REAL); EXTERN;
   {Zeichnen eines Kreises}
PROCEDURE  MOVETO  ( VAR PF : TEXT ;  X,  Y:  REAL);        EXTERN;
   {Bewegen des Zeichenstiftes ohne Zeichnen einer Linie}

{********
 *KREIS7*
 ********}
PROCEDURE  KREIS7 ( VAR  PF : TEXT ;
                       ANZ : INTEGER;:
                         X ,
                         Y ,
                         R : REAL);

   CONST       PIDRITTEL  =  1.047197551;
             ZWEIDRITTEL  =  0.66666666667;

   VAR                 N  :  INTEGER; :
                 DELTAX  , :
                 DELTAY  :  REAL;

BEGIN
   IF  ANZ  >  0
   THEN
      BEGIN
         FOR  N  :=  0  TO  5  DO          {periphere Kreis}
            BEGIN
               DELTAX  :=  ZWEIDRITTEL * R * COS ( N * PIDRITTEL );
               DELTAY  :=  ZWEIDRITTEL * R * SIN ( N * PIDRITTEL );
               KREIS7 ( PF,  ANZ-1,  DELTAX+X,  DELTAY+Y,  R / 3 );
               MOVETO ( PF,  DELTAX+X,  DELTAY+Y);
               CIRCLE ( PF,  R / 3,  0,  360)
            END  {  FOR  };
         KREIS7 ( PF,  ANZ-1 ,  X,  Y,  R / 3);  {zentrale Kreise}
         MOVETO ( PF,  X,  Y);
         CIRCLE ( PF,  R / 3,  0,  360)
      END  {  IF  }
END {  KREIS7  };
BEGIN  {  SIEBENKREISE }
   READ ( SCHICHT);
   SHEET   ( PLOT,  10,  20,  20);
   FRAME   ( PLOT);
   DELPHIN ( PLOT,  6,  2,  2);
   SIZE    ( PLOT,  1.1);
   MOVETO  ( PLOT,  18,  2);:
   MARK    ( PLOT, LRZSYMBOL );
   KREIS7  ( PLOT,  SCHICHT ,  10,  10,  9);
   MOVETO  ( PLOT,  10,  10);
   CIRCLE  ( PLOT,  9,  0,  360) {Zeichnen des Kreises fuer Schicht 0}
END  {  SIEBENKREISE  }.
```

Bild 40. Programm SIEBENKREISE mit Bibliotheksprozeduren für die Grafik und einem rekursiven Algorithmus für das Zeichen von sieben Kreisen in einem Kreis

7 Dynamische Informationsstrukturen

Bisher hat der Leser die sogenannten einfachen und strukturierten Datentypen kennengelernt. In diesem Abschnitt wird der Zeiger-Typ beschrieben, mit dem komplexe Datenstrukturen gebildet werden können.

Dazu gehören

- lineare Listen,
- Baumstrukturen,
- Datenstrukturen zur rekursiven Abarbeitung arithmetischer Ausdrücke durch einen Compiler u. a.

Diese Informationsstrukturen zeichnen sich dadurch aus, daß sie von den Eingabedaten oder von einem Algorithmus abhängen. Sie können daher nicht allein mit den einfachen und strukturierten Datentypen konstruiert werden. Diese sind bekanntlich in einem Vereinbarungsteil zu definieren und damit von statischer Natur (feste Zuordnung des Speicherplatzes). Bei ei-

nem Familienstammbaum z. B. ist die jeweilige Gesamtstruktur sicher von der betrachteten Familie abhängig; man bezeichnet sie daher als dynamisch. Der grundlegende Baustein – Eltern–Kind–Generationsfolge – einer solchen Baumstruktur ist jedoch bekannt (und statisch).

PASCAL sieht dazu den Zeiger-Typ (Pointer-Typ) vor, der an einen Datentyp fest gebunden ist. Die Vereinbarung erfolgt durch das Voranstellen eines Pfeiles (↑ bzw. ^) an den Typ bzw. an den Typ-Bezeichner (siehe dazu auch das Syntaxdiagramm in Bild 16, Abschnitt 4). Z. B. sieht die Vereinbarung einer Variablen P so aus:

P : ↑T;

T sei ein Typ-Bezeichner. Mit dem Aufruf der Standardprozedur NEW(P) wird eine dynamische Variable geschaffen, mit DISPOSE(P) wird sie gelöscht. Ein Zugriff auf eine solche Variable erfolgt durch einen Zeiger und ist daher von den anderen Zugriffsarten verschieden. *Tabelle 3* zeigt eine Zusammenstellung für die Definition und den Zugriff auf Datentypen in PASCAL (nach |21|). Der Inhalt der dynamischen Variablen ist durch den symbolischen Namen P↑ zugänglich. P↑ wird auch als Bezugsvariable – im Unterschied zur Zeiger-Variablen P – bezeichnet. Jede dynamische Variable vom Typ T hat also den gleichen Namen P↑, deshalb geschieht der Zugriff über einen Zeiger. Als eine weitere Eigenschaft läßt sich daraus ableiten, daß ihr Gültigkeitsbereich nicht von einem Anweisungsteil abhängt: sie „lebt" solange, bis sie durch ein DISPOSE(P) vernichtet wird.

Betrachtet man die Zeiger auf der Maschinenebene, so sind sie einfach Adressen. Eine Verkettung wird dadurch erzeugt, daß in einer adressierten Einheit von Speicherplätzen eine (oder mehrere) Adresse(n) für die nächste enthalten ist. Anzumerken bleibt aber, daß im Standard-PASCAL kein direkter Zugriff auf Adressen möglich ist; dies erfolgt nur durch den Compiler (sonst wären PASCAL-Programme extrem maschinenabhängig).

Für den Fall, daß ein Zeiger auf nichts mehr verweist, gibt es das Wortsymbol *nil* (Tabelle 1 im Abschnitt 2). Dieser Wert kann jedem Zeiger – also unabhängig von dem Typ, an den er gebunden ist – zugeordnet werden. Damit werden die dynamischen Datenstrukturen begrenzt.

7.1 Lineare Listen

Eine mit Zeigern verkettete lineare Liste soll einen Einstieg für das Programmieren mit Zeigern liefern. Das Anwendungsprogramm zielt darauf ab, ein Verzeichnis für das Auftreten eines Buchstabens in einem Text herzustellen. Es wäre damit ein Ausgangspunkt zu einem Programm für eine Referenzliste. Da nicht gefragt ist, wie oft ein Buchstabe, sondern in welcher Zeile er auftritt, ist die Länge der Liste unbekannt.

Zunächst Überlegungen zu dem erforderlichen Datentyp: Eine Verkettung von dynamischen Variablen erfolgt durch Zeiger, so daß ein Teil des Typs eine Referenz auf den nächsten der Liste enthalten muß. Das allein genügt natürlich nicht: es fehlt noch die Information über ein gefundenes Auftreten des gewählten Buchstabens. Da unterschiedliche Datentypen einen Baustein bilden sollen, ist ein Verbund die richtige Struktur:

```
KNOTEN = RECORD
            INHALT : INTEGER;
            NAECHSTER : KNOTENZEIGER
         END;
```

der in Anlehnung an Graphen den Namen KNOTEN erhält. Eine Komponente ist ein Zeiger (NAECHSTER) vom Typ KNOTENZEIGER, der auf den Verbund KNOTEN verweist: KNOTENZEIGER = ↑ KNOTEN. Damit stehen zwei Datentypen zur Verfügung: der Zeiger-Typ KNOTENZEIGER und der Verbund-Typ KNOTEN. Die Zeiger-Variable ZGR und die dynamische Variable ZGR↑ sind grafisch in *Bild 41* dargestellt.

Tabelle 3. Vergleich der komplexen Datentypen

Struktur	Vereinbarungs-Beispiel	Selektor	Zugriff auf eine Komponente	Komponenten-Typ
Feld	A: ARRAY[0..1000] OF REAL	A[i]	berechenbarer Index	alle identisch
Menge	S: SET OF CHAR	kein	Test auf Enthaltensein eines Elementes	alle identisch
Verbund	R: RECORD RE, IM: REAL END	R.RE	durch Komponenten-Bezeichner	koennen verschieden sein
Datei	F: FILE OF INTEGER	F^	sequentiell durch Lesen oder Schreiben	alle identisch
dynamische Daten-struktur	TYPE PTR = ^KNOTEN; KNOTEN = RECORD {Inhalt}; NEXT: PTR END; VAR P: PTR; und Erzeugung einer dyn. Variablen mit NEW(P)	kein	dyn. ueber Zeiger	

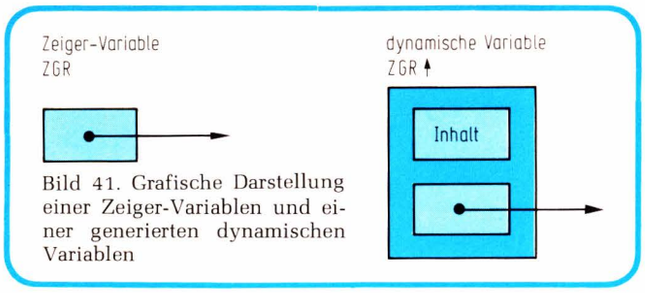


Bild 41. Grafische Darstellung einer Zeiger-Variablen und einer generierten dynamischen Variablen

Eingabe *(input)*

```
 1 PROGRAM  COPYTEXT(INPUT,  OUTPUT);
 2
 3 VAR        ZEICHEN:  CHAR;
 4
 5 BEGIN  {COPYTEXT}
 6    WHILE  NOT  EOF  DO
 7       BEGIN
 8          WHILE  NOT  EOLN  DO
 9             BEGIN
10                READ(ZEICHEN);
11                WRITE(ZEICHEN)
12             END  {WHILE};
13          READLN;
14          WRITELN
15       END  {WHILE}
16 END  {COPYTEXT}.
```

Programm

```
PROGRAM  VERZEICHNIS(INPUT,  OUTPUT);
TYPE
    KNOTENZEIGER  =  ^KNOTEN;          {Zeiger-Typ}
          KNOTEN  =  RECORD
                          INHALT:  INTEGER;
                      NAECHSTER:  KNOTENZEIGER
                     END;

VAR
          AKTUELL,
              ZGR:  KNOTENZEIGER;  {Zeiger-Variable}
                                   {      "        }
     ZEILENNUMMER:  INTEGER;
          GESUCHT,
          ZEICHEN:  CHAR;

BEGIN
   GESUCHT  :=  'R';
   ZEILENNUMMER  :=  1;
   AKTUELL  :=  NIL;
   WHILE  NOT  EOF  DO
      BEGIN
         WHILE  NOT  EOLN  DO
            BEGIN
               READ(ZEICHEN);
               IF  ZEICHEN  =  GESUCHT  THEN
                  BEGIN
                     NEW(ZGR);
                     ZGR^ .INHALT  :=  ZEILENNUMMER;
                     ZGR^ .NAECHSTER  :=  AKTUELL;
                     AKTUELL  :=  ZGR
                  END  {THEN}
            END  {WHILE};
         READLN;
         ZEILENNUMMER  :=  ZEILENNUMMER  +  1
      END  {WHILE};
   WRITELN('BUCHSTABE          ZEILEN-NUMMER(N)');
   WRITELN;
   WRITE(GESUCHT:  5,  '    ');
   REPEAT
         WRITE(AKTUELL^ .INHALT:  4);
         AKTUELL  :=  AKTUELL^ .NAECHSTER;
   UNTIL  AKTUELL  =  NIL;
   WRITELN
END  {VERZEICHNIS}.
```

Ausgabe *(output)*

```
BUCHSTABE                ZEILEN-NUMMER(N)

   R       14  13  11  10   3   3   1   1
```

Bild 42. Programm VERZEICHNIS ermittelt das Auftreten eines Buchstaben („R") in dem numerierten Programmtext COPYTEXT

Das Programm VERZEICHNIS (*Bild 42*) liest zunächst einen Text zeichenweise bis zum Dateiende ein und zählt dabei die Zeilen mit (Variable ZEILENNUMMER). Die bedingte Verzweigung enthält die Abfrage, ob das eingelesene ZEICHEN dem gesuchten Buchstaben (hier „R") gleicht. Falls dies der Fall ist, wird eine neue dynamische Variable ZGR↑ geschaffen. Ihre Komponenten erhalten Werte und der Zeiger AKTUELL (er zeigt auf die aktuelle Position der linearen Liste) wird neu gesetzt. *Bild 43* möge diese Teilschritte verdeutlichen.

Nach dem Ende des Einlesens wird eine kleine Überschrift ausgegeben. Dann erfolgt ein Rückwärtsgehen mit der Ausgabe der Komponente INHALT zum Ausgangspunkt der Kette; d. h. zu *nil*. Dies geschieht dadurch, daß der Wert des Zeigers AKTUELL den Wert der Komponenten AKTUELL.NAECHSTER erhält. Dieser verweist auf die nächstverkettete Variable.

Möchte man die Zeilennummern in gewohnter Weise in aufsteigender Reihung ausgegeben, so geht man zunächst bis zum ersten Eintrag zurück: in der „repeat"-Anweisung entfernt man den „*write*"-Aufruf. Anschließend geht der Zeiger AKTUELL so lange (mit dem „*write* (.)") vorwärts, bis er den letzten Eintrag erreicht hat. Auf diesen weist noch der unverändert gebliebene Zeiger ZGR.

In [21] findet der Leser ein Programm für die Erstellung einer Referenzliste von Namen; z. B. für die Bezeichner in einem PASCAL-Programm. Da hier auch die Anzahl der Namen unbekannt ist, wird dazu eine lineare Liste angelegt. Jedem einzelnen Namen wird dann eine eigene Liste für die Zeilennummern zugeordnet. Die Speicherung der Namen (mit dem Datentyp PACKED ARRAY [1...10] OF CHAR) erfolgt dort nach dem Prinzip der Schlüsseltransformation.

Dem aufmerksamen Leser wird im Typ-Vereinbarungsteil von VERZEICHNIS aufgefallen sein, daß der Name KNOTEN als Typ-Bezeichner bereits vor seiner Definition auftaucht. Die Vereinbarung des Verbundes KNOTEN erfolgt erst in den folgenden Zeilen. Dies ist syntaktisch richtig und die einzige für Zeiger zugelassene Ausnahme von der Regel, daß in PASCAL jeder Name *vor* seiner Verwendung vereinbart sein muß.

Bisher hatte ein Baustein der dynamischen Informationsstruktur genau eine Referenz auf den nächsten Baustein. Gibt man ihm zwei oder mehrere Verweise auf andere, so lassen sich komplexe verkettete Strukturen realisieren: binäre Bäume, Datenbanken usw.

7.2 Abschließende Bemerkungen zu Zeigern

Die feste Länge einer Zeichenkette PASCAL ist oft ein Hindernis für eine effiziente Textverarbeitung. Namen von (fast) beliebiger Länge lassen sich aber dann leicht verarbeiten, wenn man sich die Mühe macht, eine Zeichenkette STRING nicht durch ein gepacktes Feld von Zeichen (PACKED ARRAY [...] OF CHAR), sondern durch eine verkettete Liste zu realisieren. Der Baustein besteht aus einer Zeichenlänge

fester Länge – die sich am besten nach der Bitbreite des Computers richtet – und einem Verweis auf den nächsten Baustein:

```
CHUNKPTR = ↑CHUNK;
CHUNK = RECORD
          NEXT: CHUNKPTR;
          LINE: PACKED ARRAY [1..CHUNKSIZE]
                OF CHAR
        END;
```

Die Zeichenkette STRING wird je nach Anforderungen durch einen Verbund beschrieben, der mindestens einen Zeiger auf CHUNK (*chunk*: Klötzchen) enthält. Eine präzise Darstellung benötigt die Anzahl der enthaltenen Bausteine, die Gesamtanzahl der eingetragenen Zeichen, das letzte Zeichen, den Status (lesend, schreibend, nicht fertig) usw., z. B.:

```
STRING = RECORD
           W: CHAR;
           LENGTH: INTEGER;
           POSITION: 0..CHUNKSIZE;
           CHUNKNO: INTEGER;
           STATUS: (RD, WR, NO);
           START, CURRENT: CHUNKPTR
         END;
```

Mit dieser zunächst aufwendigen Konstruktion und einigen Prozeduren zum Erzeugen, Löschen, Schreiben, Lesen, Vergleichen usw. läßt sich ein Programm-Packet bilden, das vielen Anforderungen gerecht wird (z. B. für einen Text-Editor). Dieser Vorschlag ist aus [23]. Eine etwas andere Lösung findet man in [6].

Wegen einer gewissen Ähnlichkeit zu den Zeigern sollen hier noch die Puffer-Variablen erwähnt werden. Eine Puffer-Variable zeigt auf die aktuelle Position einer Datei; ihr Inhalt ist die aktuelle Komponente der Datei. Sie ist automatisch mit der Datei (wie die Bedingungen *eof* bzw. *eoln*) vereinbart; ihr Datentyp ist der Elementtyp der Datei: bei einer Textdatei *char*. *Bild 44* enthält das Programm COPYTEXT zum Kopieren einer Textdatei. Das gleichwertige Programm mit READ(.) und WRITE(.) ist die Eingabe für das Programm VERZEICHNIS in Bild 42.

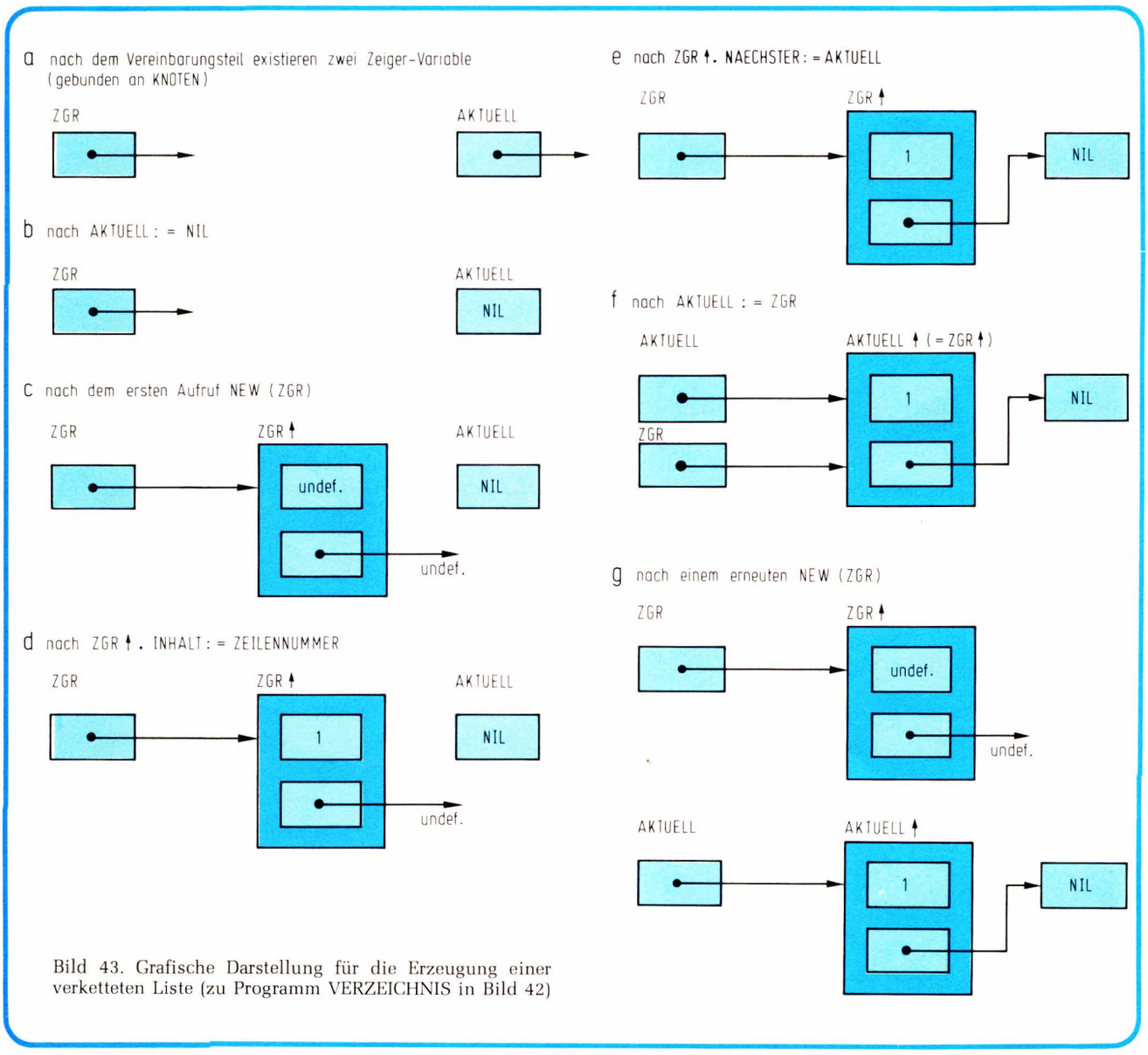


Bild 43. Grafische Darstellung für die Erzeugung einer verketteten Liste (zu Programm VERZEICHNIS in Bild 42)

8 PASCAL-Übersetzer

Während in den bisherigen Abschnitten die Sprache selbst beschrieben wurde, gilt dieser Abschnitt den PASCAL-Compilern. Aufgabe eines gut implementierten PASCAL-Systems ist, dem Benutzer ein komfortables Werkzeug zur Programm-Entwicklung zur Verfügung zu stellen. Dazu gehören u. a. für das interaktive Arbeiten am Sichtgerät ein Texteditor, ein ausgereifter Übersetzer, ein Binder für das Zuladen von Bibliotheksprogrammen und Testhilfen für die lauffähigen Programmobjekte. Anmerkung: die in dieser Serie gezeigten Beispiele sind natürlich getestet und daher ohne Übersetzungsfehler. Das kleine Programm UEBUNG enhält typische Fehler (*Bild 45*).

Eingabe *(input)*

```
Die Puffer-Variable ist in INPUT und OUTPUT genau
ein Zeichen. Definiert man dagegen z.B.:
   PROGRAM  COPYFILE(EIN,  AUS);
   TYPE     GRUNDTYP  =    .....;
   VAR      EIN, AUS  :  FILE OF GRUNDTYP;
so laesst sich in einer WHILE-Anweisung mit
   AUS^ := Ein^;  PUT(AUS);  GET(EIN);
die Datei EIN nach AUS bequem kopieren. GRUNDTYP
kann z.B. eine REAL-Groesse, aber auch z.B. ein
Verbund sein!
```

Programm

```
PROGRAM  COPYTEXT(INPUT,  OUTPUT);

BEGIN  {COPYTEXT}
   WHILE  NOT  EOF  DO
      BEGIN
         WHILE  NOT  EOLN  DO
            BEGIN
               OUTPUT^  :=  INPUT^;
               GET(INPUT);
               PUT(OUTPUT)
            END  {WHILE};
         READLN;
         WRITELN
      END  {WHILE}
END  {COPYTEXT}.
```

Ausgabe *(output)*

```
Die Puffer-Variable ist in INPUT und OUTPUT genau
ein Zeichen. Definiert man dagegen z.B.:
   PROGRAM  COPYFILE(EIN,  AUS);
   TYPE     GRUNDTYP  =    .....;
   VAR      EIN, AUS  :  FILE OF GRUNDTYP;
so laesst sich in einer WHILE-Anweisung mit
   AUS^ := Ein^;  PUT(AUS);  GET(EIN);
die Datei EIN nach AUS bequem kopieren. GRUNDTYP
kann z.B. eine REAL-Groesse, aber auch z.B. ein
Verbund sein!
```

Bild 44. Programm COPYTEXT mit den Puffer-Variablen INPUT ↑ und OUTPUNT ↑

```
     1 PROGRAM  UEBUNG(INPUT,  OUTPUT);
     2
     3 BEGIN
     4    READ(BUCHSTABE);
*****                  ^104
     5    IF  BUCHSTABE  >  'P'
*****                 ^104
     6    THEN  WRITE('P');
     7    ELSE  WRITE(BUCHSTABE)
*****       ^6
     8 END  {UEBUNG}.



COMPILER ERROR MESSAGE(S).

   6:  UNEXPECTED SYMBOL.
 104:  IDENTIFIER NOT DECLARED.
```

Bild 45. Programm UEBUNG mit Syntax-Fehler 6 wegen einem „;" vor ELSE

Der Übersetzer markiert die Position des erkannten Fehlers und hängt die Fehlernummer an, die am Ende des Programms kurz erläutert wird. Sieht man in einem Buch oder in den Syntaxdiagrammen nach, so kann die Fehlerursache meist schnell gefunden werden. Bild 15 im Abschnitt 3.4 zeigte bereits einen Fehler zur Laufzeit, der automatisch zu einem Ausdruck der (einfachen) Variablen bei Programmabbruch führt. Bei Fehlern an strukturierten Variablen sind weitere Testhilfen notwendig.

8.1 PASCAL-P

Wie bereits im Abschnitt 1 erwähnt, hat der schon fast klassisch zu nennende PASCAL-P-Compiler die Verbreitung der Sprache stark beeinflußt und auch neue Impulse für die Entwicklung von Compilern gegeben. Unter dem PASCAL-P-Compiler versteht man einen übertragbaren (portable) Übersetzer für die Sprache PASCAL, der aus einem PASCAL-Programm eine Zwischensprache – den sogenannten P-Code – erzeugt. An der ETH in Zürich entwickelt |24|, war er Ausgangspunkt für viele Implementationen von PASCAL auf Anlagen unterschiedlichster Größe.

Ausgabe I: Programmliste

```
 1        9 PROGRAM  FAKULTAET(INPUT,  OUTPUT);
 2        9
 3        9 VAR
 4        9                N:  0..  13;
 5       10              FAK:  INTEGER;
 6       11
 7       11 BEGIN
 8        3    READ(N);
 9        8    FAK  :=  1;
10       10    WHILE  N  >  1  DO
11       14       BEGIN
12       14          FAK  :=  FAK  *  N;
13       18          N  :=  N  -  1
14       19       END  {WHILE};
15       24    WRITE(FAK)
16       28 END  {FAKULTAET}.
```

Ausgabe II: P-Code

```
L   3                  ¦ BEGIN  {FAKULTAET}
 ENT   1   L   4
 ENT   2   L   5
 LAO           9       ¦    READ(N)
 LDA   0       5
 CSP         RDI
 LDCI          1       ¦    FAK:=1
 SROI         10
L   6                  ¦    WHILE
I   10
 LDOI          9       ¦         N>1 DO
 LDCI          1
 GRTI
 FJP       L   7
 LDOI         10       ¦       FAK:=FAK*N
 LDOI          9
 MPI
 SROI         10
 LDOI          9       ¦       N:=N-1
 LDCI          1
I   20
 SBI
 CHKI  0      13       ¦       Test auf  0<=N<=13
 SROI          9
 UJP       L   6
L   7                  ¦    END {WHILE}
 LDOI         10       ¦    WRITE(FAK)
 LDCI         10       ¦        {Laenge: 10 Zeichen}
 LDA   0       6
 CSP         WRI
 RETP                  ¦ {Rueckkehr vom Hauptprogramm}
L   4=        11
L   5=         8       ¦ END {FAKULTAET}
Q
I    0
 MST           0       ¦ Aufruf des Hauptprogramms
                       ¦   FAKULTAET als Prozedur
 CUP   0   L   3
 STP                   ¦ Stop
Q
```

Bild 46. Ausgabe des Programms FAKULTAET nach der Übersetzung mit dem PASCAL-P-Compiler (oben) und des P-Codes mit Erläuterungen (unten).

Kennzeichnend – und zunächst überraschend – für diesen Compiler ist, daß er in PASCAL selbst geschrieben ist. In dieser Form als Quelle ist er natürlich noch nicht lauffähig, d. h. er kann noch keine Programme übersetzen. Erst durch einen Ur-Übersetzer (bzw. Interpreter) des P-Codes (s. u.) erhält man einen ersten lauffähigen Compiler. Der P-Compiler – mit ca. 4000 Zeilen ein großes Programm – ist selbst nach der Methode der schrittweisen Verfeinerung geschrieben und umfaßt die folgenden Sequenzen:

1. Syntax-Analyse,
2. Syntax-Analyse mit Behandlung kontextfreier Fehler,
3. Erzeugung der Compilertabellen,
4. Behandlung kontextbehafteter Fehler,
5. Adreßzuweisung an die Variablen,
6. Erzeugung des Codes.

Die beiden Punkte 5 und 6 lassen sich leicht einer speziellen Maschine anpassen. Daher kommt man relativ schnell zu „seinem" Compiler, ohne die großen Teile 1...4 selbst entwickeln zu müssen. Der interessierte Leser möge sich in [24, 25] näher informieren ([25] enthält eine Aufsatzserie über den P-Compiler). Im weiteren soll der P-Code näher betrachtet werden.

Der P-Code kann als die Assemblersprache eines hypothetischen, stapelorientierten Computers (engl.: *stack computer*, im folgenden mit SC abgekürzt) aufgefaßt werden. Charakteristisch für einen solchen Computer ist, daß alle Operationen auf dem Stapel (auch mit Keller bezeichnet) stattfinden. Einige Taschenrechner (z. B. die der Firma Hewlett-Packard) arbeiten nach diesem Prinzip; siehe daher auch den Beitrag über die Programmiertechnik IPS [26]. Da die meisten Mikroprozessoren einen Stapel im Hauptspeicher direkt verwalten (mit PUSH- und POP-Befehlen), ist die Anpassung an den Befehlssatz der Zielmaschine vereinfacht.

Der SC besteht aus einem linear adressierbaren Speicher und fünf Registern:

PC Programm-Zähler,
SP Zeiger auf die Spitze des Stapels,
MP Zeiger auf die Parameter-Deskriptoren,
NP Zeiger auf das aktuelle Datensegment,
EP Zeiger auf den maximal benutzten Speicher.

Ein Teil des Speichers enthält den P-Code des geladenen Programms, der andere die statischen Variablen im Stapel und die dynamischen Variablen (durch NEW (...) erzeugt) in der Halde, engl. *heap*. Stapel und Halde sind meist so organisiert, daß sie aufeinander zuwachsen, und, falls sie sich berühren, zu einem Langzeitfehler wegen Speichermangel führen. Ein Assemblerbefehl des SC enthält die Komponenten: einen Operations-Code und bei Bedarf die Parameter P und Q. P steht meist für die Differenz der Blöcke zwischen Aufruf und Vereinbarung einer Variablen. Q dagegen gibt dann die relative Verschiebung zur Basis MP an. Das Datensegment enthält die Information über die aufzurufende Prozedur (bzw. Funktion): die Rückkehradresse, Parameter u. a.

Das kurze Programm FAKULTAET soll als Beispiel für Anwendung des PASCAL-P-Compilers dienen. *Bild 46* zeigt die Ausgabe des Programms nach der Übersetzung und den erzeugten P-Code. Eine Assembler-Anweisung des SC ist wahlweise eine Zeilenzählung (I...), eine Marke (L...) oder ein Assemblerbefehl (beginnend mit einem Leerzeichen). Der vierte Buchstabe eines Befehls gibt den Datentyp an: *integer:* I, *real:* R, *boolean:* B, *char:* C, *procedure:* P, *set:* S und Adresse: A. Die *Tabelle 4* enthält einen Ausschnitt der Assemblerbefehle. Die für Assemblersprachen typischen Sprünge (hier FJP und UJP) formen das klare Programm FAKULTAET in einen „Spaghetticode".

Für den P-Code existiert ein Interpreter-Programm [24], das den erzeugten Assemblertext einliest, assembliert und schließlich ausführt. Mit diesem Interpreter erhält man nach einer Eingabe für N den Wert von N!. Allerdings benötigt der Interpreter um den Faktor 10 mehr Rechenzeit als der direkte Compiler (und Lader) auf der gleichen Rechenanlage. Für Produktionsprogramm ist also die Ausführung über die Zwischensprache mit anschließender Interpretierung langsam. Dagegen wäre eine Anwendung des P-Codes auf einem Mikrocomputer denkbar, wenn eine größere Anlage die PASCAL-Programme übersetzt und er die einfachere Interpretation übernimmt. Kürzere Rechenzeiten erhält man, wenn der ursprüngliche P-Compiler in einen speziellen Compiler für den Mikrocomputer geändert wird.

Auf der Basis des besprochenen P-Codes wurde der SC auf dem Einplatinen-Computer EUROCOM-II (Hersteller: Eltec Elektronik GmbH, Mainz) verwirklicht. In der ersten lauffähigen Version benötigte der Interpreter auf der M6809-Maschine 12 Sekunden zur Berechnung der Funktion *Ackermann* (3, 4) [20]. In der ELEKTRONIK erscheint dazu ein Aufsatz.

8.2 Kommerzielle PASCAL-Compiler

In diesem Sonderheft findet der Leser einen Aufsatz über kommerzielle PASCAL-Systeme (Welches PASCAL für welchen Computer?). Ohne die dort aufgeführten Produkte im einzelnen zu bewerten, muß angemerkt werden, daß nicht immer ein Standard-PASCAL geboten wird. Ein potentieller Benutzer wird es aber zu schätzen wissen, wenn er das PASCAL nach *Jensen/Wirth* [1] vorfindet und dann dazu natürlich die „Extras" des Herstellers.

Tabelle 4. P-Code Befehlsliste (Auszug)

Befehl	Parameter T	P	Q	Beschreibung
ADI				ganzzahlige Addition
CHK	x	x	x	Test auf Bereichsgrenzen
CSP			x	Aufruf einer Standardprozedur
CUP		x	x	Aufruf einer benutzer-definierten Prozedur
DEC			x	Subtrahiere Q
ENT			x	Prozedureingang
EQU	x		(x)	Test auf "gleich"
FJP			x	FALSE-Sprung nach Q
GRT	x		(x)	Test auf "groesser als"
INC			x	Addiere Q
IXA			x	Berechne indizierte Adresse
LAO			x	Lade Adresse der Hauptprogramm-Variablen
LDA		x	x	Lade Adresse der Variablen
LDC	x		x	Lade Konstante vom Typ T
LDO	x		x	Lade Hauptprogramm-Variable
LOD	x	x	x	Lade Variable vom Typ T
MPI				ganzzahlige Multiplikation
MST		x		Einleitung einer Aufrufsequenz
SBI				ganzzahlige Subtraktion
SRO	x		x	Speichere in die Hauptprogramm-Variable
STP				Halt
STR	x	x	x	Speichere in die Variable
RET	x			Prozedurausgang
UJP			x	Unbedingter Sprung nach Q
UNI				Vereinigung von Mengen

Erwähnenswert von den genannten Produkten ist die Pascal-MIKROENGINE von Western-Digital. Dieser Rechner besitzt einen Hardware-Decodierer zur direkten Ausführung des P-Codes. Dieser P-Code ist auf eine 16-Bit-Architektur zugeschnitten und weicht von dem im letzten Abschnitt beschriebenen Code ab. Damit ist sicherlich der Weg für neue Computer-Architekturen gezeigt: nämlich, daß sich das Gerät nach den Erfordernissen der Sprache richtet und nicht wie bisher umgekehrt. Das UCSD-PASCAL (*University of California, San Diego*) ist ebenfalls bemerkenswert. Dieses Programmpaket umfaßt nicht nur den Compiler, sondern auch ein komplettes Floppy-Disk-Betriebssystem, eine Programm-Bibliothek, einen Text-Editor, einen Programm-Binder u. a. mehr. Der Benutzer arbeitet daher mit einem eigenständigen System; alle Programme dazu sind in PASCAL erstellt. Von einer amerikanischen Firma (Sof Tech, San Diego) ist das UCSD--Pascal in Lizenz erhältlich. In diesem Zusammenhang ist auf |27| hinzuweisen. Dieses – für Anfänger geschriebene – Buch kann nur demjenigen empfohlen werden, der eine UCSD-PASCAL-Version mit einem Grafik-Bildschirm benutzt (die Abweichungen vom Standard sind zum Teil erheblich).

8.3 Erweiterungen

Neben den schon im Abschnitt 6.4 angesprochenen Änderungen (Prozedur-Parameter und dynamische Feldgrenzen), bleiben dem geplanten internationalen Standard vor allem folgende Schwächen, die unterschiedliche Lösungen gefunden haben:

- Wertzuweisungen an Felder, Mengen und Verbunde (zusätzlicher *value*-Vereinbarungsteil, z. B. PASCAL 6000 von CDC),
- indexsequentielle Dateien (gibt es z. B. im Mikroprozessor PASCAL von TI),
- Datentyp STRING (im UCSD-PASCAL mit wählbarer Länge, z. B. NAME |15|).

Für die Einbindung von Assembler-Code hat es sich am vorteilhaftesten erwiesen, den Übersetzer mit Nicht-Standard-Prozeduren (bzw. Funktionen) zu erweitern (z. B. die Prozedur TRAP im Z80-PASCAL von Zilog). Dies gilt auch für das Fehlen des Exponentations-Operators.

Eine wesentliche Verbesserung der Programmentwicklung ist durch neue interaktive Editier- und Test-Methoden zu erwarten. Das heißt z. B.: der Übersetzer unterbricht beim Auftreten eines Fehlers seine Verarbeitung und ermöglicht dem Benutzer eine sofortige Korrektur. Ein Arbeiten im „Dialog" mit PASCAL wird dann auch verwöhnte BASIC-Programmierer zum Umsteigen veranlassen.

In dieser Arbeit stand die Behandlung des sequentiellen PASCAL im Vordergrund. Das Standard-PASCAL hat trotz aller Diskussionen ein abgeschlossenes Konzept. Mit seiner konsequenten Strukturierung der Anweisungen und Daten hat PASCAL seinen Platz unter den Programmiersprachen. Weiterentwicklungen betreffen das wichtige Gebiet der Verarbeitung paralleler Prozesse (Betriebssysteme u. a.). Neben dem bereits zitierten Concurrent-PASCAL |10| und MODULA-2 |11| ist in dem Zusammenhang noch PORTAL zu nennen |28|.

Die hier dargestellten Programmbeispiele wurden auf der Rechenanlage *Cyber 175* (Control Data Corp.) des Leibnitz-Rechenzentrums der Bayerischen Akademie der Wissenschaften, München, erstellt. Der Verfasser dankt den Studenten *W. Axmann* und *F. Zinoni* für ihre Mitarbeit an einigen Programmen sowie seiner Frau für ihre hilfreiche Unterstützung.

Literatur

1 *Jensen, K.; Wirth, N.*: PASCAL. User Manual and Report. 2. Aufl., Springer-Verlag, New York, Heidelberg, Berlin, 1978.

2 *Wirth, N.*: The Programming Language Pascal. Acta informatica, Bd. 1 (1971), S. 35...63.

3 *Wirth, N.*: Revidierter Bericht über die Programmiersprache PASCAL. Akademie-Verlag Berlin, 1976.

4 *Hoare, C.A.R.; Wirth, N.*: An Axiomatic Definition of the Programming Language PASCAL. Acta Informatica, Bd. 2 (1973), S. 335...355.

5 *Kaucher, E.; Klatte, R.; Ullrich, Ch.*: Höhere Programmiersprachen ALGOL, FORTRAN, PASCAL. BI-Wissenschaftsverlag, Reihe Informatik, Bd. 24, Mannheim 1978.

6 *Rohlfing, H.*: PASCAL. Eine Einführung. BI-Hochschultaschenbücher, Bd. 756, Mannheim, 1978.

7 *Schauer, H.*: PASCAL für Anfänger. 3. Aufl., Oldenbourg Verlag Wien München 1979.

8 *Herschel, R.; Pieper, F.*: PASCAL. Oldenbourg Verlag, München Wien, 1979.

9 *Wilson, I. R.; Addyman, M.*: PASCAL. Hanser Verlag München, 1979.

10 *Hansen, P. Brinch*: The Programming Language Concurrent Pascal. IEEE Transactions on Software Engineering, SE-1 (1975), H. 2, S. 199...207.

11 *Wirth, N.*: MODULA-2. Berichte des Instituts für Informatik, Nr. 36, ETH Zürich, 1980.

12 *Condiet, M. N. et al.*: PASCAL PROGRAM FORMATTER. Pascal News, Nr. 13, December 1978, S. 45...58.

13 *Joepgen, H. G.*: Komfort-Programm zum Rechnen mit komplexen Zahlen. ELEKTRONIK 1979, H. 17, S. 53...58.

14 *Koch, G. R.*: Systematisches Softwareengineering für Mikrocomputer. 1. Teil. ELEKTRONIK 1979, H. 21, S. 49...56.

15 *Mende, Simon*: Physik. Gleichungen und Tabellen. München 1976.

16 *Davis, W. D. T.*: Systemerkennung für adaptive Regelsysteme. Oldenbourg Verlag, München 1973.

17 *Jordan-Engeln, G., Reutter, F.*: Numerische Mathematik für Ingenieure. BI-Hochschultaschenbücher, Bd. 104, Mannheim, 1978.

18 *Zemanek, H.*: Über den Kalender, Teil 1 und 2. Elektronische Rechenanlagen 20 (1978) H. 6, S. 263...268; 21 (1979) H. 4, S. 165...170 und H. 6, S. 265...267.

19 *Bulirsch, R., Stoer, J.*: Numerical Treatment of Ordinary Differential Equaltions by Extrapolation Methods. Num. Mathematik, Bd. 8 (1966), S. 1...13.

20 *Dorfner, F.*: PASCAL. ELEKTRONIK 1980, H. 3, S. 43...46 und ELEKTRONIK-Sonderheft „PASCAL".

21 *Wirth, N.*: Algorithmen und Datenstrukturen. Teubner Verlag, Stuttgart, 1975.

22 *Lehnert, R.*: Musizieren mit Licht. Bild der Wissenschaften 12 (1975), H. 10, S. 144...150.

23 *Bishop, J. M.*: Implementing Strings in Pascal. Software-Practice and Experience (1979) Bd. 9, S. 779...788.

24 *Nori, K. V. et al.*: The PASCAL 'P' Compiler: Implementation Notes. ETH-Zürich, 1976.

25 *Davis, Henry*: The Pascal Notebook. INTERFACE AGE 1980, H. 7 und folgende Hefte.

26 *Meinzer, K.*: IPS – eine neue Programmiertechnik für Mikrocomputer. ELEKTRONIK 1978, H. 15, S. 35...42.

27 *Bowles, K. L.*: Microcomputer Problem Solving using PASCAL. Springer-Verlag, New York, 1977.

28 *Lienhard, H.*: Die Echtzeitprogrammiersprache PORTAL. Eine Übersicht Landis & Gyr-Mitteilungen (1978), Bd. 25, H. 2, S. 2...8.

Dipl.-Ing. Eberhard Enger stammt aus Wiener Neustadt. Er studierte zunächst am Ohm-Polytechnikum in Nürnberg und dann an der TU München. Nach einer Tätigkeit als freier Mitarbeiter ging er an das Zoologische Institut der Universität München und entwikkelt dort Geräte und Programme für die Meßwerterfassung und -verarbeitung, daneben wissenschaftliche Mitarbeit an Forschungsvorhaben der Biokybernetik und Lehrauftrag für das Fach Bioinformatik.
Privattelefon: (0 89) 39 08 46
Hobbys: Astronomie, Bergwandern, Skilaufen
ELEKTRONIK-Leser seit 1966

Elektronik Sonderheft III zum Thema Mikroprozessoren

Produkte – Anwendungen – Tendenzen

Das Sonderheft III der ELEKTRONIK über Mikroprozessoren ist die Folge-Publikation der inzwischen vergriffenen Sonderhefte I und II, hat aber daraus nur die immer wieder zur Orientierung und zur Festigung der Terminologie gerne gelesene Einführungsarbeit aus Heft I übernommen, diesmal aber in gründlich überarbeiteter Form, der heutigen Situation angepaßt.

Es folgen in übersichtlicher, systematischer Zusammenstellung 16 inzwischen in der ELEKTRONIK erschienene Arbeiten, die den fortgeschrittenen Mikroprozessor- und Mikrocomputer-Anwender ansprechen. Kern dieser Serie ist vor allem die Vorstellung der 16-Bit-Prozessoren 8086, MC68000, Z8000 und 9440, die bekanntlich eine neue Ära in der „Mikrocomputerei" einleiten und die sorgfältig studiert und verglichen sein wollen, bevor man sich für einen von ihnen entscheidet.

Zwei neue Arbeiten über das Editieren und Assemblieren sowie über einen universellen Cross-Assembler wurden speziell für den Software-Mann hinzugefügt.

Alles in allem spiegelt Sonderheft III sehr klar die Weiterentwicklung auf diesem Sektor wider:

- Neue Produkte bieten mehr Möglichkeiten: Lohnt es sich, sie einzusetzen?
- Welchen Aufwand erfordert die Anwendung?
- Welchen Weg beschreitet die µP-Technik? Wie wird man in Zukunft die Software erstellen?

Das sind die aktuellen Fragen, die Entwickler und Manager gleichermaßen interessieren – ja interessieren müssen. Genau diese Themenkreise behandelt das neue Sonderheft. Es erspart mühevolles Nachschlagen in zahlreichen Einzelheften, aber auch das Sammeln von Firmenunterlagen und Fotokopien. Am Schluß des Heftes ist ein umfangreiches Tabellenwerk mit den Anschriften der Hersteller- und Vertriebsfirmen aufgeführt.

Inhalt 136 Seiten, Preis 18.– DM

Bestellung

Sie erhalten dieses Heft bei allen Bahnhofsbuchhandlungen, Elektronik-Bauteilehändlern, größeren Buchhandlungen oder direkt beim FRANZIS-Verlag.

Bitte haben Sie Verständnis, daß der Verlag Einzelhefte aus organisatorischen Gründen nur gegen Vorauszahlung liefern kann. Wir bitten Sie, in diesem Fall als Bestellung den Betrag von 19.50 DM (18.– DM plus 1.50 DM Porto) auf unser Postscheckkonto (München Nr. 813 75-809) mit Hinweis ELEKTRONIK Sonderheft III, Mikroprozessoren – Produkte – Anwendungen – Tendenzen – zu überweisen. Bitte vergessen Sie nicht, auf dem Zahlungsbeleg in Druckschrift Ihre volle Anschrift anzugeben. Sofort nach Eingang der Zahlung senden wir Ihnen das Heft zu.

Franzis-Verlag
Karlstraße 37, 8000 München 2,
Telefon 0 89/51 17-2 39

Das Heft erhalten Sie auch in der Schweiz beim Verlag Thali AG, CH-6285 Hitzkirch und in Österreich beim Fachbuch Center Erb, A-1061 Wien, Amerlingstraße 1

Dipl.-Ing. Franz-Josef Dorfner

Ein Mikrocomputer – für PASCAL konzipiert

An der Kalifornischen Universität von San Diego wurde von Kenneth Bowles sowie seinen Mitarbeitern das Wirthsche PASCAL modifiziert und erweitert, was unter der Bezeichnung UCSD-PASCAL bekannt geworden ist. Diese UCSD-Version ist mit besonderer Berücksichtigung der Praxisorientiertheit und der Implementierbarkeit auf den heutigen Mikrorechnern entwickelt worden. Mittlerweile ist PASCAL schon auf nahezu allen gängigen Rechnern bzw. Prozessoren implementiert worden. Die technisch bemerkenswerteste und für den Anwender auch leistungsfähigste Implementation der Programmiersprache PASCAL gelang der US-Firma Western Digital in Form der „PASCAL-MICROENGINE". Diese PASCAL-Maschine, eingebaut in ein Gehäuse mit zwei 8-Zoll-Floppy-Disk-Drives und Stromversorgung wird beschrieben. Unter der Bezeichnung SE-90 wird sie von der Firma Spezial-Electronic vertrieben.

1 P-Code-Maschine

Wenn auch in den letzten Jahren durch die Verwendung von höheren Programmiersprachen, wie z. B. PASCAL, die Softwareerstellung für den Programmierer immer komfortabler wurde, so hat sich auf der untersten Rechnerebene bisher nichts geändert. Heutige Rechner und auch noch solche in absehbarer Zukunft arbeiten immer noch nach der zweiwertigen Logik. Das heißt, die arithmetisch-logische Einheit (ALU) eines Prozessors kennt nur die Zustände „Low" oder „High" bzw. „0" oder „1".

Um ein in PASCAL geschriebenes Programm dem Rechner verständlich zu machen, muß dafür gesorgt werden, daß das PASCAL-Source-Programm (Quellentext) in eine für den Rechner verständliche Eins-Null-Darstellung umgesetzt wird. Diese Umsetzung wird von einem Compiler durchgeführt. Dabei sind natürlich mehr oder weniger effektive Lösungsmöglichkeiten denkbar. (Siehe hierzu *Bild 1...4*).

Dabei spielt auch die Portabilität eine große Rolle. Portable Programme sind auf Rechnern mit unterschiedlichen Befehlssätzen lauffähig. Die gemeinsame Ebene, auf der die Programme ausgetauscht werden können, ist ein Zwischencode. Im Falle von PASCAL ist dies der sogenannte P-Code. Er ist ein optimaler Instruktionssatz für die Ausführung von PASCAL-Programmen. Er beinhaltet Befehle, die direkt den im PASCAL-Programm ausgeführten Funktionen entsprechen.

Der P-Code ist so optimiert, daß die am häufigsten benutzten PASCAL-Operationen den geringsten Speicheraufwand benötigen und mit der größtmöglichen Geschwindigkeit ausgeführt werden können. Zum Beispiel bestehen alle P-Code-Instruktionen aus nur einem Byte und für die meisten Zugriffe auf Datenfelder wird nur ein zusätzliches Byte als Zeiger benötigt. Soll beispielsweise ein PASCAL-Programm auf dem Prozessor Z80 laufen, so muß der P-Code softwaremäßig interpretiert werden, oder eine weitere

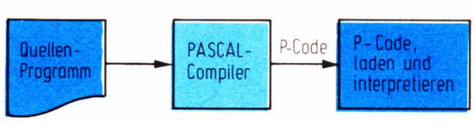


Bild 1. Die meisten in Zusammenhang mit Mikroprozessoren benutzten PASCAL-Compiler arbeiten interpretativ: Der PASCAL-Compiler generiert den P-Code, der dann über einen Software-Interpreter in den Maschinencode des jeweiligen Prozessors übersetzt wird. Die Interpretation jeder P-Code-Instruktion benötigt in der Regel mehrere Prozessorinstruktionen

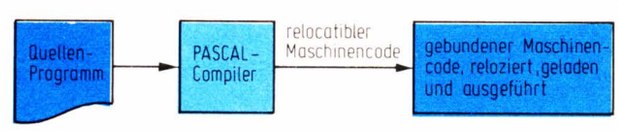


Bild 2. Andere PASCAL-Compiler übersetzen die PASCAL-Quellenprogramme direkt in den Maschinencode des jeweiligen Prozessors. Damit laufen die PASCAL-Programme auf einer Rechnerarchitektur, die für diese Sprache nicht effizient ist. Daraus resultiert eine große Zahl von notwendigen Maschineninstruktionen

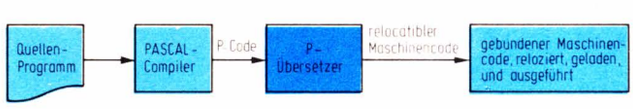


Bild 3. Ein PASCAL-P-Code-Übersetzer wandelt den vorher compilierten P-Code in den Maschinencode des Zielprozessors um. Jede P-Code-Instruktion wird in einen oder mehrere Maschinenbefehle übersetzt

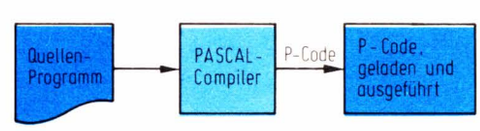


Bild 4. Die PASCAL-MICROENGINE benutzt keines der in Bild 1...3 geschilderten Verfahren. Der PASCAL-Compiler übersetzt das Quellenprogramm in den P-Code, den der Prozessor als seine spezifische Maschinensprache direkt ausführt

Umsetzung erzeugt aus dem P-Code den prozessorspezifischen Maschinencode.

Diese Vorgehensweise entspricht der „klassischen" Methode. Das heißt, die Implementierung der Sprache erfolgt auf einem schon existenten, von seiner Architektur her nicht unbedingt optimalen Rechner. Man kann auch überspitzt sagen, die Sprache wird auf eine bestehende Hardwarekonfiguration gezwängt.

Bei der „PASCAL MICROENGINE" wurde ein völlig neuartiges Systemkonzept verwirklicht. Hier wurde erstmalig die Hardware an die Software angepaßt und damit an die speziellen Belange der Programmiersprache PASCAL. Western Digital realisierte hardwaremäßig die sogenannte Pseudo-P-Code-Maschine. Der P-Code ist bei dem 16-Bit-Prozessor dieser Maschine per Firmware in drei ROMs abgelegt. Damit ist der Befehlssatz dieses Prozessors identisch mit dem P-Code, d. h. die MICROENGINE ist mikroprogrammiert.

Der P-Code kann hier vom Prozessor direkt ausgeführt werden. Es ist keine softwaremäßige Interpretation oder eine weitere Umsetzung notwendig. Daraus resultiert eine schnellere Programmausführung gegenüber herkömmlichen Implementationen. Der Geschwindigkeitsvorteil liegt bei ca. fünf- bis zwanzigfach.

2 Hardware

Berücksichtigt man nicht das Floppy-Disk-System und die Stromversorgung, so ist der Computer SE-90 auf einer Platine von nur 8 x 16 Zoll untergebracht (*Bild 5*). Diese Single-Board-Version wird auch von Spezial-Electronic angeboten und eignet sich besonders für OEM-Anwendungen.

Der Kern des PASCAL-Computers ist der Prozessor WD 9000 von Western Digital. Dieser stackorientierte 16-Bit-MOS/LSI-Prozessor ist ein Chipsatz, der die in der höheren Programmiersprache PASCAL geschriebenen Programme direkt ausführt. Der Chipset besteht aus fünf jeweils vierzigpoligen Chips:

- Arithmetik-Baustein
  enthält die arithmetisch-logische Einheit, die Mikroinstruktionsdecodierung und den Registerfile,
- Steuerbaustein
  enthält den Mikroinstruktionszähler, die Ablaufsteuerung und die Ein-/Ausgabe-Steuerlogik,
- drei „Mikroms"
  jedes ROM ist 512 x 22 Bit organisiert, sie enthalten die Mikroinstruktionen.

Der Prozessor wird von einem Vierphasentakt mit einer Frequenz von 3 MHz getaktet. Für die Takterzeugung hat Western Digital den Baustein WD 2143 entwickelt.

Der Prozessor verfügt über eine Floating-Point-Hardware sowie über eine hardwaremäßige Multiplikation und Division (Instruktionen für 16-Bit-Multiplikation und 16-Bit-Division). Die I/O-Ports sind Memory-Mapped, d. h. ein I/O-Port wird wie eine Speicherzelle behandelt. Die Programmierung dieser I/O-Ports ist in der höheren Programmiersprache PASCAL möglich (siehe hierzu das Programmbeispiel in *Bild 6*). Der Prozessor verfügt auch über die beiden Befehle „Signal" und „Wait on Semaphore". Damit können Programmsynchronisierungen realisiert werden.

Der PASCAL-Computer SE-90 verfügt bereits über 64 KByte dynamischen RAM-Speicher mit Refreshlogik. Es stehen zwei serielle V-24-Schnittstellen mit getrennt einstellbarer Übertragungsrate von 110...19 200 Baud zur Verfügung. Als Schnittstellencontroller ist der Baustein WD 1931 eingesetzt, der sowohl asyn-

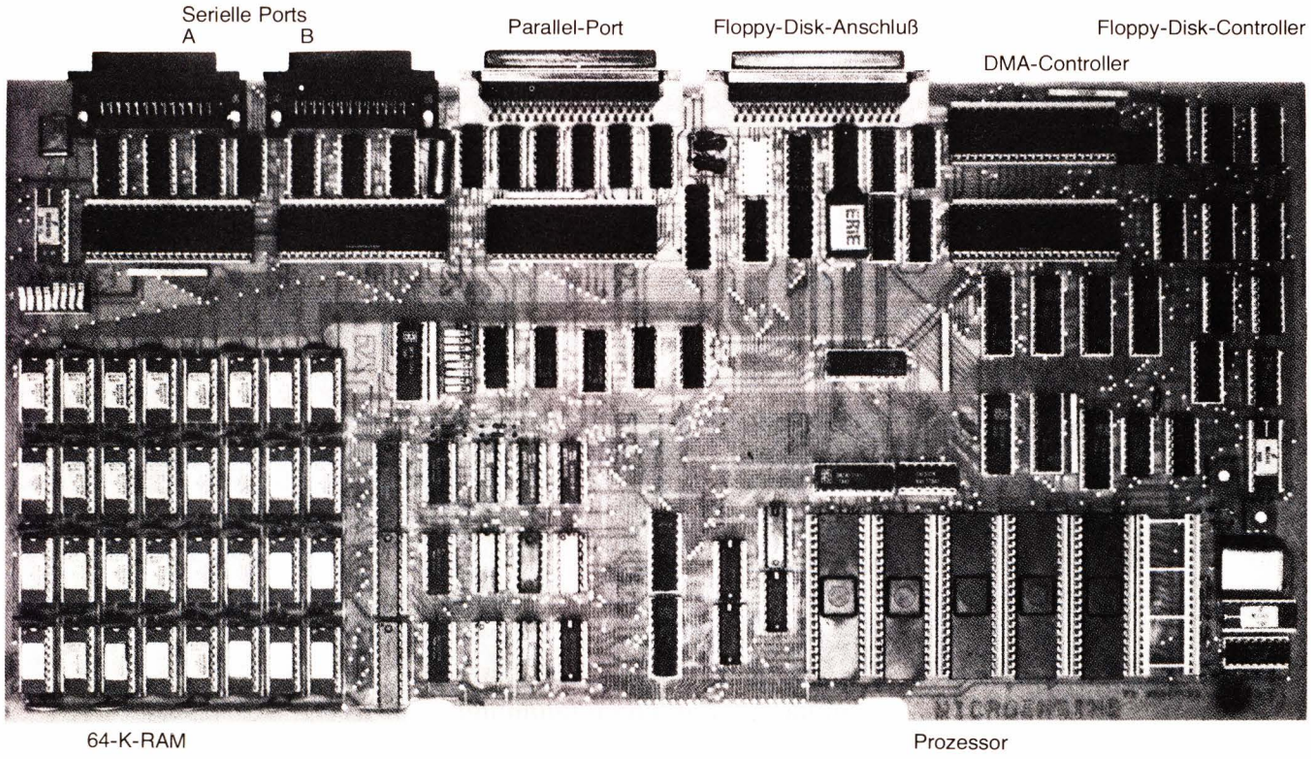


Bild 5. Die Computerplatine enthält alle Bauelemente der MICROENGINE

chrone wie auch synchrone Datenübertragung ermöglicht. Pinkompatibel zu diesem Baustein ist der WD 1933, mit dem auch bei entsprechender Treibersoftware die Datenübertragungsprotokolle SDLC, HDLC oder ADCCP realisiert werden können.

Zur Verfügung steht auch noch ein Parallel-Port in der Form des programmierbaren Interface-Bausteins 8255A. Dieser hat drei 8 Bit breite Ports, die individuell programmiert werden können. Die Voreinstellung dieser Ports erfolgt beim Laden des Betriebssystems und zwar: Port A zu einem 8-Bit-Eingangsport, Port B zu einem 8-Bit-Ausgangsport und das Port C als Kontrollport (Handshake-Leitungen). Es kann dann am Parallel-Port jeder Drucker mit einer Centronics-Standardschnittstelle betrieben werden. Ein Drucker mit serieller Schnittstelle kann an das serielle Port angeschlossen werden. Auf dem Computer-Board befindet sich auch die gesamte Logik für den Floppy-Disk-Anschluß. Verwendet werden die beiden Bausteine FD 1791 als Floppy-Disk-Controller und der DM 1883 als Direct-Memory-Access-Controller.

Es können bis zu vier Floppy-Antriebe desselben Typs angeschlossen werden. Der Floppy-Disk-Controller ist per Schalter auf einfache oder doppelte Schreibdichte und auf 8-Zoll- oder $5^1/_4$-Zoll-Laufwerke einstellbar. Es können sowohl einseitige wie auch doppelseitige Floppy-Systeme verwendet werden. Das System SE-90 ist mit zwei 8-Zoll-Floppy-Disk-Laufwerken ausgestattet, wobei zwischen einseitigen und doppelseitigen ausgewählt werden kann.

3 Software

Lauffähig auf dem Computersystem SE-90 ist die Programmiersprache PASCAL in der Version III.0 von der Universität von San Diego mit dem UCSD-Betriebssystem. Über die Programmiersprache PASCAL sollen hier keine Aussagen gemacht werden (siehe hierzu die Aufsätze in diesem Sonderheft). Es soll hier nur über das UCSD-Betriebssystem berichtet werden. Bemerkenswert ist, daß das gesamte Betriebssystem in PASCAL geschrieben ist.

Das UCSD-Betriebssystem ist hierarchisch aufgebaut und arbeitet nach der Menütechnik. Das heißt, es ist ein interaktives System. Dem Benutzer wird stets eine Vorschlagzeile auf der Konsole ausgegeben, die ihn über die möglichen Kommandos entsprechend der hierarchischen Ebene im Betriebssystem informieren. Nach der Initialisierungsphase meldet sich das System in der höchsten Kommandoebene. Durch Eingabe eines Kennbuchstaben können die folgenden Moduln des Betriebssystems aktiviert werden:

- Bildschirmorientierter Editor
- File-Manager
- PASCAL-Compiler
- Library
- Linker
- Eine Reihe von Unterstützungsprogrammen.

Der bildschirmorientierte Editor bietet dem Anwender den größten Komfort. Das Betriebssystem verfügt zusätzlich über zwei weitere Editoren, den zeilenorientierten Editor (aufzurufen mit „YALOE") und einen Editor für sehr große Files, z. B. für die Textbearbeitung (aufzurufen mit „L2").

Der File-Manager ist für die Dateienverwaltung zuständig. Der Basisfile des Betriebssystems ist der Workfile. Dieser ist eine Kopie des gerade bearbeiteten Files und hat auf der Diskette den Namen „SYSTEM.WRK.TEXT", wenn es ein Textfile ist, oder „SYSTEM.WRK.CODE", wenn es ein Codefile ist. Der PASCAL-Compiler führt die Umsetzung des PASCAL-Source-Programmes in den P-Code aus. Normalerweise wird der Workfile compiliert. Ist kein Workfile vorhanden, so fragt das System, welcher File compiliert werden soll. Der UCSD-PASCAL-Compiler bietet die Möglichkeit, Compileroptionen, wie z. B. Erstellen eines Compilerlistings, anzugeben.

Das UCSD-PASCAL unterstützt die separate Compilierbarkeit. Mit dem Linker können Programmteile gebunden werden. Werden in einem zu erstellenden Programm Routinen aus der System-Library verwendet, so wird bei der Compilation automatisch der Linker aufgerufen. Das Betriebssystem hat Unterstützungsprogramme wie: Disketten-Formatierungsprogramm, Setup-Programm für die Anpassung verschiedener Terminals an das System, Calculator-Programm, womit die PASCAL-Maschine im Tischrechnermodus arbeitet, Disassembler und weitere.

4 Entwicklungsziele

Das System SE-90 ist zukunftsorientiert und wird auch neueren Entwicklungen gerecht. Zur Zeit ist ein Hard-Disk-Anschluß mit einer 20-MByte-Platte in Entwicklung. Ebenso arbeitet man in Verbindung mit der 20-MByte-Platte an einem Mehrplatzsystem.

Für Anwender, die glauben, ohne BASIC nicht auszukommen, wird etwa Herbst 1980 ein BASIC-Interpreter verfügbar sein.

Literatur

1 *Western Digital:* Handbuch „PASCAL MICROENGINE".

a

```
program io_write;      (* i/o-programmierung *)

const  port_b = -992; (* adresse des seriellen ports b *)
           z = 65;    (* zu sendendes Zeichen 'A' *)

var          x : record case boolean of
                 true: (a:integer);
                 false:(p:^integer);
                 end;
begin
  x.a:=port_b;
  x.p^:=z;
end.
```

b

```
                SEGMENT=  1  PROCEDURE=   1  BLOCK=  1   BLOCK OFFSET=   2
EXIT IC:  0013                   DATA SEGMENT SIZE:  0001
                 block #  1   offset in block=   6
SEG PROC     OFFSET                                  HEX CODE
  1    1     0(000):   SLDC          6                06
  1    1     1(001):   LSL           0                9900
  1    1     3(003):   SPR                            D1
  1    1     4(004):   LDCI       -992                8120FC
  1    1     7(007):   STL           1                A401
  1    1     9(009):   SLDL          1                20
  1    1    10(00A):   LDCB         65                8041
  1    1    12(00C):   STO                            C4
  1    1    13(00D):   CXG      BLKEXIT               940318
  1    1    16(010):   RPU           1                9601
```

Bild 6. In diesem Programmbeispiel wird ein E/A-Port programmiert. Das Quellenprogramm (a) ist in (b) disassembliert im P-Code (in mnemonischer und sedezimaler Darstellung)

Dipl.-Ing. Heinz Kitzhöfer

System-PASCAL

Programmiersprache mit abgerundetem Konzept

Nachdem in der ELEKTRONIK die Programmiersprache PASCAL bereits ausführlich beschrieben wurde, sollen hier die speziellen Gesichtspunkte dargestellt werden, die zur Entwicklung des System-PASCAL geführt haben. Diese Sprache hebt sich durch das abgerundete Konzept sowie besondere Eingeschaften von anderen PASCAL-Erweiterungen ab.

Wie muß ein modernes Softwarewerkzeug aussehen?

Einerseits gezwungen durch Kostensteigerung im Software-Bereich andererseits angeregt durch neue Programmier- und Sprach-Konzepte aus dem Bereich der Informatik suchte die Firma Dietz schon vor Jahren nach einem modernen Werkzeug, das sowohl die System- und Standardsoftware-Entwicklungsgruppen im eigenen Hause als auch einen größeren Kundenkreis in die Lage versetzen sollte, effektiver zu arbeiten.

Dabei war zu berücksichtigen, daß das schwächste Glied einer Kette alle Verbesserungen an den übrigen Stellen eines Systems nutzlos machen kann; so muß eine erfolgreiche Optimierung im Bereich der Software auf alle Lebensphasen eines Programms Einfluß nehmen – von der Systemanalyse über den Entwurf, die Implementierung, den Test, die Installation bis hin zur Wartung und Pflege und eventuellen Erweiterungen und Änderungen.

Das bedeutet, eine moderne Programmiersprache muß anwendungsnahe Strukturen besitzen, einen Top-Down-Entwurf begünstigen, und sie muß leicht lesbar und möglichst selbstdokumentierend sein.

Da der Compiler ja selbst auch ein Software-Produkt ist, muß auch er in einer Sprache implementiert sein, die diese Eigenschaften besitzt. Ideal ist es, wenn der Compiler in seiner eigenen Sprache geschrieben ist, denn dann muß nicht mehr die Programmiersprache *und* eine Compilersprache gepflegt werden, sondern beide Sprachen sind identisch.

PASCAL – die geeignete Gastsprache

Am Ende dieser Überlegungen unter den oben genannten Gesichtspunkten fiel die Wahl auf PASCAL. Diese Programmiersprache ist zwar nicht so gängig wie z. B. FORTRAN, ALGOL, COBOL oder BASIC, aber sie besitzt im Gegensatz zu diesen Sprachen fast alle Eigenschaften, die die gestellten Anforderungen erfüllen.

In der Tat verfügt die von *Prof. Niklaus Wirth* entworfene Sprache über einen außerordentlich wirksamen, aber zugleich sinnvoll begrenzten Umfang an Kontrollstrukturen, Datentypen, Datenstrukturen und anderen Funktionen, welche die effiziente Erstellung von Programmen nahezu erzwingen.

Nachdem die Vorzüge dieser Sprache schon weitgehend dargestellt worden sind, soll hier gezeigt werden, wie auf PASCAL ein universelles Software-Werkzeug aufgebaut worden ist.

Ein optimaler PASCAL-Compiler

Der erste Schritt mußte natürlich sein, einen PASCAL-Compiler zu erstellen, der die Strukturen und Funktionen der Sprache PASCAL optimal auf die Hardware der Rechner-Familie DIETZ-621 umsetzte. Der Compiler-Entwurf und die anschließende Implementierung wurde an der Universität Kiel am Lehrstuhl für Informatik von *Prof. Langmaack* durchgeführt.

Die hohe Effizienz des von diesem PASCAL-Compiler erzeugten Maschinencodes zeigt sich darin, daß im Vergleich zu guter Assembler-Programmierung in bezug auf Laufzeit und Codelänge ein Faktor von durchschnittlich nur 1,5 auftritt; das ist um so bemerkenswerter, da diese Tests an betriebssystemnahen Moduln durchgeführt wurden. Damit steht der Einsatz von PASCAL auch in zeitkritischen Prozeß-Anwendungen offen. In der Vergangenheit mußte unter anderem wegen der Anforderungen an die Verarbeitungsgeschwindigkeit und dem Speicherbedarf im

Bereich der Prozeßsteuerung und der Echtzeit-Verarbeitung häufig die hardwarenahe Programmierung im Assembler erfolgen.

Die wachsende Komplexität der zu erstellenden Software, die wachsende Bedeutung der Pflegekosten gegenüber dem Erstellungsaufwand und die erhöhten Anforderungen an die Zuverlässigkeit verlangen jedoch eine eher problemorientierte als maschinennahe Programmierung.

Das erweiterte PASCAL-Programmiersystem

Zum Einsatz in diesen Problembereichen mußte ein Weg führen, der die Erfüllung folgender Forderungen gewährleistet:

- Über den im Dietz-PASCAL-Compiler implementierten Sprachumfang sollten keine Erweiterungen der Sprache erfolgen,
  - um die Einfachheit und Klarheit der Sprache zu erhalten,
  - und um dem Gesichtspunkt der Kompatibilität Rechnung zu tragen und einer künftigen Normung der Sprache nicht entgegenzuarbeiten.
- Das Verfahren, die genannten Anwendungsbereiche einem PASCAL-Programmiersystem zu erschließen, sollte auch auf weitere Anwendungsbereiche übertragbar sein.

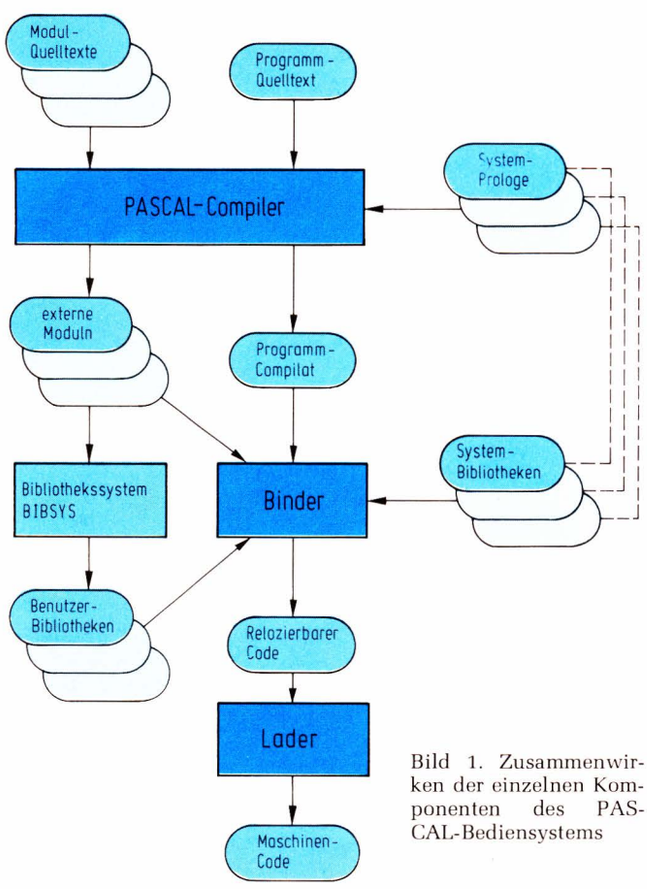


Bild 1. Zusammenwirken der einzelnen Komponenten des PASCAL-Bediensystems

- Das PASCAL-Programmiersystem selbst sollte nicht mit dem Erschließen dieser Anwendungsbereiche belastet werden,
  - um dieses Programmiersystem einfach zu halten,
  - und um eine Entbündelung bezüglich unterschiedlicher Anwendungsbereiche zu ermöglichen.

Das Konzept von System-PASCAL

Es wurde deshalb folgendes Konzept implementiert, das bei Dietz unter dem Namen System-PASCAL bekannt ist:

- je Anwendungsbereich wird ein Satz von Funktionen in einer Bibliothek bereitgestellt.
- Diese Funktionen sind mit den Mitteln der Sprache PASCAL aufrufbar. Dazu wird auf das – auch von *Prof. Wirth* vorgeschlagene – Mittel der „externen Prozeduren" zurückgegriffen.
- Diese Prozeduren werden dem Programmiersystem über einen „Prolog" als (Quasi-)Standardprozeduren bekannt gemacht.

Kurz: System-PASCAL ist eine Erweiterung des Anwendungsspektrums durch die Aufrufbarkeit von problemorientierten, externen Prozeduren mit den Mitteln der Sprache PASCAL.

Ein wesentliches Merkmal dieses Konzeptes ist die Tatsache, daß alle Prüfungen, die zur Übersetzungszeit vom Compiler für PASCAL-Standard-Funktionen durchgeführt werden können, auch für die System-PASCAL-Prozeduren gemacht werden können. Das heißt im Klartext, daß zum Beispiel Parameterprüfungen zur Laufzeit komplett entfallen.

Die Komponenten des PASCAL-Bediensystems

Bild 1 zeigt das Zusammenwirken der einzelnen Komponenten des PASCAL-Bediensystems. Der Compiler erzeugt in gleicher Weise den Code eines abgeschlossenen PASCAL-Programms, wie auch den eines externen Moduls, das jeweils aus einer oder mehreren Prozeduren bestehen kann. Deklarationen, die anwendungsspezifisch geordnet sind, erhält der Compiler dabei automatisch aus einem System-Prolog.

Externe Moduln können entweder als Einzeldateien gehalten werden oder über das Bibliothekssystem BIBSYS zu Benutzer-Bibliotheken zusammengefaßt werden. Die Benutzer-Bibliotheken können auch vom Anwender definierte Assembler-Routinen enthalten.

Daß darüber hinaus die Bibliotheken sprachunabhängig sind und damit von allen bei Dietz verfügbaren Compilern benutzt werden können, liegt in der internen Struktur des Gesamtprogrammiersystems begründet.

Der Binder sammelt alle für ein spezielles PASCAL-Programm notwendigen Elemente – wie Prozeduren, Funktionen und Operatoren – aus der anwendungsspezifischen System-Bibliothek (die eindeutig mit dem entsprechenden System-Prolog verknüpft ist), den externen Moduln und den Benutzer-Bibliotheken. Der Binder unterstützt Overlay-Definition in Baumstruktur, wobei die Baumzweige dynamisch während der Laufzeit auf- und abgebaut werden.

Der entstandene relozierbare Code wird durch den Lader wählbar vorab oder bei jedem Start des Programms durch Adreß-Substitution in absoluten Maschinen-Code gewandelt.

Die Funktionsgruppen des Programmiersystems System-PASCAL

Der Einsatz von System-PASCAL in unterschiedlichen Anwendungsbereichen ist durch das Anbinden von Basis-Funktionen ermöglicht, die wie schon erwähnt über die Schnittstelle der „PASCAL-procedure" und der „PASCAL-function" mit der Programmiersprache verknüpft sind. Diese Basis-Funktionen lassen sich in zwei Gruppen einteilen, die einmal den Bereich der Standard-Betriebssystem-Funktionen umfassen, die allen Sprachen direkt zugänglich sind, und zum anderen aus komplexen Moduln bestehen, die speziell auf das System-PASCAL hin entwickelt oder zumindest an dieses angepaßt worden sind.
Bild 2 zeigt grafisch beide Funktionsgruppen.

Funktionen des Betriebssystems

Zu den direkt verfügbaren Betriebssystemfunktionen gehören

- der sequentielle und direkte File-Zugriff über den Modul FIAC (File Access),
- die Nutzung der außerordentlich leistungsfähigen SPOOL-Funktionen.
- Über die „normalen" Eigenschaften eines SPOOL hinaus können SPOOL-Aufträge selektiert, modifiziert und in ihren Eigenschaften verändert, gelöscht oder nach Abbruch neu aufgesetzt werden. Änderbare Eigenschaften sind u. a. Priorität, Formularart, lang- oder kurzfristige Sicherung von Daten. Ferner kann die Abarbeitung solcher Aufträge, die nach Priorität und Formularart gesteuert ist, unterbrochen, fortgesetzt und wiederholt werden. Die Daten können für weitere SPOOL-Aufträge erhalten bleiben. Der SPOOL-Betrieb kann sofort oder verzögert beendet werden.
- Die Einbettung in das Rechnernetz-System DIXOS. DIXOS verbindet Rechner der Familie DIETZ-621 zu Netzwerken, die physikalisch beliebig vermascht werden können, und die von allen Rechnern – und damit von allen Benutzern – Zugriff auf die Betriebsmittel des Gesamtsystems gewähren. Zu den verfügbaren Betriebsmitteln gehören Peripheriegeräte, Dateien, aber auch Rechenkapazität. Die Anwenderprogramme sind unabhängig davon, ob sie in einer Einrechnerkonfiguration oder in einem Netz eingesetzt werden.
- Das Betriebssystem DIXOS arbeitet zum Teil dynamisch, d. h. bei Ausfall eines Betriebsmittels kann auf ein anderes, gleichartiges Betriebsmittel umgeschaltet werden, das sich an einem anderen Punkt des Rechnernetzes befindet. Ferner können der Verwaltung des Betriebssystem DIXOS Betriebsmittel entzogen oder bekanntgemacht werden.
- Die Bedienung aller in der Produktpalette enthaltenen DFÜ-Pakete, wie Dialog-Prozeduren, RJE-Emulationen und X.25-Protokolle verschiedenen Typs und unterschiedlichen Niveaus für den Anschluß an andere EDV-Systeme und öffentliche Netze.

Erst auf Grund der Ergänzung durch die zweite Gruppe der Funktionen wird System-PASCAL das, was es ist: *das* Werkzeug.

Dieser Teil beinhaltet

- ein komplexes Datenhaltungssystem (DFMS),
- ein umfassendes Prozeß-Steuerungssystem (REALTIME) und
- ein System zur Unterstützung der Dialogverarbeitung am Bildschirm.

Das komplexe Datenhaltungssystem DFMS

Seit 1975 ist im kommerziellen Einsatzbereich des C-BASIC-Programmiersystems der Modul DFMS (*Disk File Management System*) erprobt. Aufgrund seiner Stabilität und seiner funktionellen Eigenschaften wurde er mittlerweile auch im COBOL-Programmiersystem eingesetzt und schließlich auch in das PASCAL-System einbezogen.

Mit DFMS lassen sich problemangepaßte Datenstrukturen auf einem Plattenspeicher anlegen und

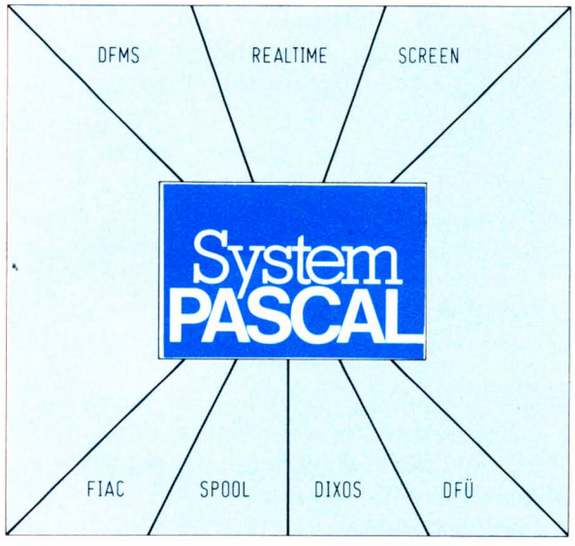


Bild 2. Zur Zeit verfügbare Funktionsgruppen des offenen Programmiersystems „System-PASCAL"

verwalten. Zur Verfügbarkeit der Funktionen von DFMS wurde ein Satz von Prozeduren geschaffen, der eine Kopplung zwischen PASCAL-Programm und dem ins Betriebssystem integrierten Modul DFMS darstellt, so daß

- diese Prozeduren mit den Mitteln der Sprache aufrufbar sind und
- die Sprache selbst sowie der PASCAL-Compiler nicht angetastet werden.

Die wichtigsten Merkmale des so für PASCAL-Programme erschlossenen Datenhaltungssystems DFMS sind:

- Es stehen unterschiedliche Zugriffsfunktionen zu Plattendateien zur Verfügung:
  - der direkte Zugriff über eine (logische) Satznummer,
  - der sequentielle Zugriff,
  - der indexdirekte und
  - der indexsequentielle Zugriff über Schlüssel.
- Je Satzdatei können mehrere Schlüsseldateien angelegt werden, so daß auf die Sätze der Datei nach unterschiedlichen Sortierkriterien zugegriffen werden kann.
- Schlüssel können ein- und mehrdeutig definiert werden.
- Suchbegriffe werden mit den Vergleichsoperatoren <, < =, =, > = und > verknüpft.
- Zum Sortieren, Invertieren und Reorganisieren von Dateien stehen Dienstprogramme zur Verfügung.

Mit dieser Einbettung des Datenhaltungssystems in das PASCAL-Programmiersystem ist komfortables Daten-Handling im technisch-wissenschaftlichen Bereich sowie auch im Bereich der Echtzeit-Datenverarbeitung ermöglicht.

Das umfassende Echtzeit-System

Um die Funktionen zur Echtzeit-Datenverarbeitung in das PASCAL-System einzubeziehen, standen zwei alternative Vorgehensweisen zur Auswahl: Schaffung von Prozeduren zur Ankopplung an vorhandene Funktionen des Betriebssystems oder eine unmittelbare Realisierung der notwendigen Funktionen in dem zu erstellenden Satz von Prozeduren.

Der ersten Alternative widersprachen folgende Gründe:

- Man hätte einen Verlust an Laufzeiteffizienz hinnehmen müssen, was für zeitkritische Prozesse problematisch geworden wäre.
- Die im Betriebssystem XOS verfügbaren Echtzeit-Funktionen sind auf die aus FORTRAN, PEARL oder BASEX kommenden Anforderungen zugeschnitten, andererseits fehlen aber wirksame und zugleich einfache Mechanismen zur Kommunikation und Synchronisation von Prozessen.

Es wurde daher ein Satz von Prozeduren entwickelt, der keine Kopplung zum Betriebssystem darstellt, sondern in dem die für Echtzeit-Verarbeitung notwendigen Funktionen unmittelbar vorhanden sind und die mit den Mitteln der Sprache PASCAL aufrufbar sind.

Dabei mußten Verfahren zur Kommunikation und Synchronisation paralleler Prozesse entwickelt sowie eine Möglichkeit des Zeitschedulings geschaffen werden. Die wesentlichen Eigenschaften dieses Prozeß-Systems sind folgende:

- Die Anzahl parallel arbeitsfähiger Prozesse ist variabel. Bis zum Ausbau des Systems auf 12 Prozesse kann die Hardware-Struktur der 621-Systeme von Dietz optimal genutzt werden.
- Zur Prozeß-Kommunikation und Prozeß-Synchronisation ist das Konzept der Nachrichten (Messages) realisiert. Dieses Konzept ist einfach zu verstehen, aber gleichzeitig auch mächtig und wirkungsvoll in der Anwendung:
  - Prozesse senden und empfangen Nachrichten. Damit kann man eine zeitliche Synchronisation mit gleichzeitigem Datenaustausch (Inhalt einer Nachricht) erreichen.
  - Prozesse bestimmen ihre Priorität selbst.
  - Prozesse sind „abgeschlossene" PASCAL-Programme.

Mit diesem „Message"-System sind die Einzelprogramme zu einem Gesamtsystem konkurrierender, sich gegenseitig synchronisierender und miteinander kommunizierender Prozesse verknüpft.

- Einer der Prozesse kann als „Zeitprozeß" ausgezeichnet sein, der also das Scheduling der übrigen Prozesse übernimmt. Auch dieser „Zeitprozeß" ist über das Message-System erreichbar und kann beauftragt werden:
  - einmalig zu einem bestimmten Zeitpunkt
  - oder n-mal in einem gewählten Zeitraster Nachrichten an einen Prozeß zu übermitteln.
- Es stehen Funktionen zur Interrupt-Behandlung zur Verfügung, die Interrupts erkennen, die Interrupt-Quelle selektieren und über Daten des Typs Menge die Interrupt-Prozesse (auch PASCAL-Programme) mit den notwendigen Parametern versorgen. Die besonders effiziente Implementierung des Datentyps Menge und der Mengenoperationen im PASCAL-System ermöglichen Reaktionszeiten für einen Interrupt-Prozeß, die im µs-Bereich liegen.
- Ferner stehen Funktionen zur Verfügung, die bitweisen, wortweisen, binären und dezimalen Zugriff auf digitale und analoge Peripherie gestatten.

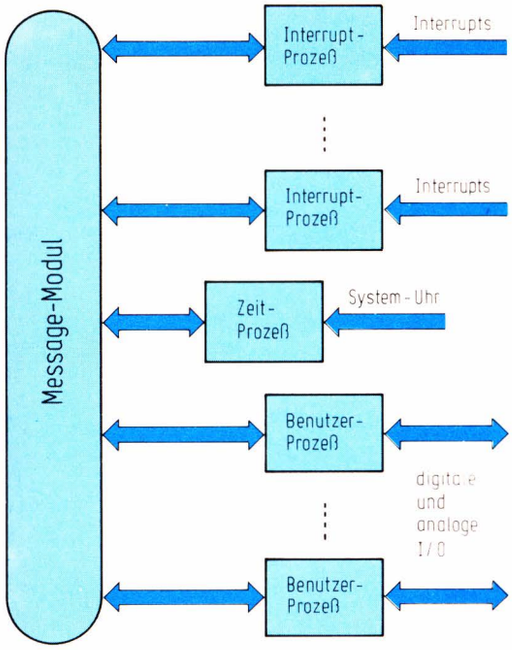


Bild 3. Peripherie- und Intertask-Kommunikation der konkurrierenden PASCAL-Prozesse

Mit dieser Realisierung eines Echtzeit-Systems ergänzen sich

- die sehr guten Eigenschaften der Programmiersprache PASCAL und
- die effiziente Implementierung des PASCAL-Systems auf den Dietz-Computern und
- das konzeptionell einfache, aber mächtige Instrument der „Messages" der Echtzeit-Datenverarbeitung

zu einem Werkzeug, mit dem komplexe Problemlösungen für Hersteller und Anwender gleichermaßen kostengünstig entwickelt und gepflegt werden können.

Unterstützung der Dialog-Verarbeitung am Bildschirm

Mit dem Produkt „SCREEN MANAGEMENT" ist das PASCAL-Programmiersystem nach demselben Verfahren wie im Fall Echtzeit-System um einen Satz von Prozeduren ergänzt worden, der die Dialog-Verarbeitung über Bildschirm unabhängig vom Gerätetyp unterstützt. Die untere Ausbaustufe dieses SCREEN MANAGEMENT stellt dem PASCAL-Programmierer die Basis-Funktionen eines komfortablen Bildschirmgerätes zur Verfügung. Die zweite Ausbaustufe erlaubt echte Dialog-Verarbeitung über Masken mit Ein- und Ausgabe von Daten und Plausibilitäts-Prüfungen auf eingegebene Daten. Ferner werden Funktionen zur Definition und zur Pflege von Masken bereitgestellt. Schon während dieser Phase kann auf ein Grundgerüst zum System gehörender Masken zurückgegriffen werden.

Die wesentliche Eigenschaft dieses Produktes ist die Führung des Bedieners in seiner Anwendung. Gerade in kommerziellen Anwendungssystemen bestehen die höchsten Anforderungen an solche dialoggestützte Datenverarbeitung. Da das Produkt diesen Anforderungen voll entspricht, sind auch im technisch-wissenschaftlichen Bereich vor allem im Zusammenhang mit einem komplexen Datenhaltungssystem ausgezeichnete Einsatzvoraussetzungen für dieses Produkt gegeben.

Schlußbemerkungen

Zusammenfassend sollen noch einmal drei Punkte hervorgehoben werden.

1) Das PASCAL-System, bestehend aus PASCAL-Compiler und PASCAL-Laufzeitroutinen ist eingebettet in ein PASCAL-Programmiersystem, das aus den zusätzlichen Komponenten EDITOR, BINDER und LADER besteht. Das PASCAL-Programmiersystem wiederum ist eingebettet in ein Gesamt-System mit den Komponenten Bediensystem und Betriebssystem, das die effiziente Nutzung aller Betriebsmittel erlaubt und sowohl die Erstellung als auch die Ausführung von PASCAL-Programmen hervorragend unterstützt.

2) Das PASCAL-Programmiersystem ist so realisiert, daß es mit den erwähnten Verfahren funktional erweiterbar ist.
Die funktionale Erweiterbarkeit des PASCAL-Programmiersystems gestattet, zusätzliche Anwendungsbereiche zu erschließen, ohne die Sprache oder den Rest des Programmiersystems zu ändern. Damit ist ein *offenes* Programmiersystem geschaffen.
Neben dem technisch-wissenschaftlichen Anwendungsbereich ist der Bereich der Echtzeit-Datenverarbeitung erschlossen. Für alle Anwendungsbereiche sind umfangreiche Möglichkeiten zur Verwaltung großer Datenbestände und zur Dialog-Datenverarbeitung vorhanden.

3) System-PASCAL ist keine neue oder eine Änderung einer vorhandenen Sprache – mit der damit verbundenen Gefahr der Zerstörung dieser Sprache –, sondern eine reale *Methode*, PASCAL ein breites Anwendungsspektrum zu erschließen.

Heinz Kitzhöfer studierte an der RWTH Aachen Elektrotechnik. Seine praktische Tätigkeit begann 1972 in der Betriebssystem-Entwicklung der Firma Dietz. Danach konzipierte und entwickelte er ein Rechnersystem am Rechenzentrum der Universität Hohenheim. 1978 wechselte er wieder zur Industrie über und ist jetzt Leiter des Supports im Bereich der Technischen Computer-Systeme bei der Firma Dietz.
Telefon: (0 71 21) 8 23 33
Hobbys: Heimwerken, Wochenendgrundstück.
ELEKTRONIK-Leser seit 1972.

Dipl.-Ing. Eberhard Enger

Welches PASCAL für welchen Computer?

Seit etwa drei Jahren findet die Programmiersprache PASCAL – bis dahin auf den Hochschulbereich beschränkt – verstärkt Anwendung im industriellen Bereich. Diese lange Anlaufzeit erklärt sich zum Teil damit, daß PASCAL-Übersetzer zunächst nur auf großen Rechenanlagen verfügbar waren. Die bekannten Fortschritte in der Mikroprozessor- und Speichertechnologie und der PASCAL-P-Compiler der ETH Zürich führten dazu, daß mittlerweile PASCAL auch für viele Mikrocomputer erhältlich ist. In dieser Übersicht sind deshalb nur PASCAL-Systeme für Mini- und Mikrocomputer enthalten, Versionen für Großrechner sind schon seit längerem bekannt.

Voraussetzung für ein bequemes Arbeiten mit PASCAL auf einem Mikrocomputer ist ein schneller Massenspeicher (Diskette, Magnetplatte u. ä.) sowie ein Speicher von ca. 48...64 KByte Größe. Ein übergeordnetes Betriebssystem muß einen Texteditor für die Bearbeitung des PASCAL-Quellenprogramms, eine Verwaltung der Dateien, eine Steuerung der peripheren Geräte (Sichtgerät usw.) und ein Analyse-Programm für aufgetretene Fehler in der Ausführungsphase enthalten. Fast durchgängig möglich ist das Einbinden von Bibliotheks-Programmen und von Assembler-Unterprogrammen.

Die Tabelle enthält eine Spalte für die Verarbeitung des PASCAL-Textes zu einem lauffähigen (Maschinen-) Programm. Dafür existieren zwei Möglichkeiten:

a) Übersetzung in einen sogenannten P-Code und anschließender Interpretation Q → P, I;

b) direkte Übersetzung in den Maschinen-Code und dann unmittelbare Ausführung Q → M.

Im Fall b) gibt es eine Variante mit dem Assembler-Code oder, benannt nach der Programmiersprache C, mit der C-Code-Zwischenstufe. Die Methode a) führt in der Regel zu einer längeren Laufzeit, benötigt aber weniger Speicherplatz als b). Ein Optimum ist sicher erreicht, wenn der P-Code direkt durch einen mikroprogrammierten Prozessor interpretiert wird. Dies geschieht z. B. bei der WD 9000 von Western Digital und bei der MK 16 von Mikros Systems.

Neben diesen Fortschritten, die zu einer schnelleren Bearbeitung von Programmen führen, sind die Anwender jetzt besonders an nützlichen und fertigen Programmen interessiert. Einige Hersteller bieten bereits hier etwas an (z. B. zur Textverarbeitung).

Weil die internationale Standardisierung der Programmiersprache PASCAL noch nicht abgeschlossen ist, unterscheiden sich die aufgeführten PASCAL-Übersetzer. Nach wie vor wird jedoch das Buch PASCAL, User Manual and Report von *K. Jensen* und *N. Wirth* (1978) als geltender Standard akzeptiert.

Übersicht PASCAL-Versionen verschiedener Anbieter

Hersteller/Anbieter	Bezeichnung	Computer-Modell	CPU	Verarbeitung[1])	Betriebssystem	Bemerkung
Advanced Micro Devices	PASCAL	AmSYS 8/8	AmZ8000, Z80 Am9080A/8080A Am8085A	Q→C→M	CP/M	
Alpha Micro	UCSD-Pascal	AM-100	16 Bit		CP/M, UCSD-POS[2])	
AMI Microsystems	AMI PASCAL	8002A (Tektronix), EXORciser, MDC	S6800, Z80 S9900, PDP-11 S2200			
Apple Computer	Apple Pascal Apple Pascal	Apple II Apple III	6502 6502	Q→P, I	UCSD-POS	
BSP Thomas Krug	PASCAL/M PASCAL/Z PASCAL PLUS	CP/M-kompatibel CP/M-kompatibel CP/M-kompatibel	– – –	Q→P, I Q→A→M R→P, I	CP/M CP/M CP/M	
Columbia Data Products	USCD-Pascal	Commander	Z80A	Q→P, I	CP/M, USCD-POS	

Hersteller/Anbieter	Bezeichnung	Computer-Modell	CPU	Verar-beitung[1])	Betriebs system	Bermerkung
Commodore		3032	6502			(4. Quartal 1980)
Compucorp	PASCAL	625	Z80	Q→M	ZEBRA	
Creative Daten Systeme	PASCAL	Data-General-Rechner	–	Q→M	AOS, MP/OS	
Data General	MP/PASCAL	MP/100, MP/200, ECLIPSE	microNOVA	Q→M	MP/OS, AOS	Aufrufe von Betriebs-funktionen
Diehl Datensysteme (TA)	PASCAL-P4	alphatronic dds 2[cs], 3[cs]	8085 16 Bit		DOS 20	(4. Quartal 1980)
Dietz Computer-Systeme	System-Pascal	DIETZ 621	16 Bit	Q→M	TSOS	Echtzeit-verarbeitung
Digital Equipment		VAX-11 PDP-11, LSI-11	32 Bit 16 Bit			Vertrieb durch DECUS
Exidy	UCSD-Pascal	Sorcerer	Z80			
FMG Corp.	PASCAL/MT PASCAL/M	CP/M-kompatibel, TRS-80	8080, 8085, Z80 8080, 8085, Z80	Q→M Q→P, I	CP/M CP/M	
FELTRON Elektronik	PASCAL/M PASCAL/MT	FELTRON 5080 FELTRON 5080	8085 8085	Q→P, I Q→M	MP/M MP/M	
Hewlett Packard	Pascal/1000 Pascal/64000	HP 1000 HP 64000	16 Bit (8080, 8085 Z80 u. a.)	Q→A→M Q→M	RTE –	PASCAL-Compiler für den Ziel-prozessor
Intel	PASCAL-80 PASCAL-86	MDS-800, Intellec II 221, 231, 241	8080, 8085 8086	Q→P, I Q→M	ISIS II ISIS	(4. Quartal 1980)
ITT		ITT 2020	6502			(3. Quartal 1980)
Kontron	PASCAL	PSI ψ 80	Z80A	Q→P, I	KOS	
Lucidata	P-6800	FLEX-kompatibel	M6800, 6809		FLEX	
Mikros Systems		Mk-16	16 Bit	Q→P		
Motorola	M6809 PASCAL I PASCAL	EXORciser II EXORmacs	MC6809 MC68000	Q→P→M Q→M	MDOS VERSADOS	
MPDV	UCSD-Pascal PASCAL/MT	M2000-D M2000-D	Z80 Z80	Q→P, I Q→M	CP/M CP/M	
OHIO Scientific	UCSD-Pascal	Challenger	6502			(3. Quartal)
OMSI	OMSI PASCAL V1. 2	DEC-Rechner	–	Q→A→M	RT-11, RSX, IAS, RSTS	
Ramtek	UCSD-Pascal	Modell 6114	Z80	Q→P	RAMOS	Farbgrafik
rowi	UCSD-Pascal	microwi 65	6502	Q→P, I	MONA	
SALOTA	UCSD-Pascal	MFC 512	Z80A	Q→P, I	SALDOS	
Siemens	PASCAL (BS2000) PASCAL 300	7000 6610 6620 6640 300	32 Bit 8 Bit 16 Bit 16 Bit 16 Bit	Q→M Q→M Q→P, I Q→M Q→M	BS2000 BS1 BS2 BS3 ORG	
Sorcim	PASCAL/M		8080, 8085 Z80	Q→M	CP/M	

Hersteller/Anbieter	Bezeichnung	Computer-Modell	CPU	Verar-beitung[1])	Betriebs-system	Bermerkung
Spezial-Electronic	UCSD-Pascal	PASCAL P-CODE MASCHINE (SE-90)	16 Bit (WD9000 von Western Digital)	Q→P	UCSD-POS	direkte Aus-führung des P-Codes
SWT	UCSD-PASCAL	SWT-6809	6809	Q→P, I	UCSD-POS	
Tandy	UCSD-Pascal, PASCAL/MT	TRS-80	Z80			siehe: FMG INTERFACE AGE Verlagsges.
Texas Instruments	Mikroprozessor Pascal (MPP) TI PASCAL	TMS 9900 TMS 9900	FS990/AMPL, DS990	Q→P→M Q→O	TX5 DX10, TX990 RX990	Echtzeitver-arbeitung
Western Digital Corp.	UCSD-Pascal	WD 90 (PASCAL-MICRO-ENGINE)	16 Bit (WD9000)	Q→P	UCSD-POS	direkte Aus-führung des P-Codes
Zilog	Z80 PASCAL Z80 PASCAL	MCZ, ZDS	Z80 Z8000	Q→P, I	RIO	(3.Quartal 1980)

[1]) Erläuterung der Zeichen: Q = Pascal-Quellprogramm; P = P-Code; M = Maschinen-Code; A = Assembler-Code; C = C-Code; I = P-Code-Interpreter
[2]) UCSD-POS: UCSD PASCAL OPERATING SYSTEM (UCSD ist die Abkürzung für: *University of California, San Diego*)
Die Tabellen wurden aufgrund von Unterlagen der Hersteller bzw. Anbieter zusammengestellt und erheben keinen Anspruch auf Vollständigkeit.

Anschriften der Firmen (Vertrieb)

Advanced Micro Devices GmbH, Rosenheimer Str. 139, 8000 München 80, Tel. (0 89) 40 19 76

Alpha Micro (digitronic computersysteme GmbH, Am Kamp 17, 2081 Holm (Holstein), Tel. (0 41 03) 8 86 72)

AMI (mtc mikrotec gmbh, Rosenbergstr. 94, 7000 Stuttgart 1, Tel. (07 11) 61 10 36)

Apple Computer Marketing Gesellschaft mbH, Überseering 25, 2000 Hamburg 60, Tel. (0 40) 6 30 80 45

BSP Thomas Krug EDV-Beratung, Postf. 11–44, 8400 Regensburg, Tel. (09 41) 9 39 45

Columbia Data Products (Ziegler-Instruments GmbH & Co. KG, Postfach 510, 4050 Mönchengladbach, Tel. (0 21 66) 8 00 91)

Commodore GmbH, Frankfurter Str. 171–175, 6078 Neu-Isenburg, Tel. (0 61 02) 80 03

Compucorp (Beagrand Datentechnik GmbH & Co., Berliner Str. 2–6, 6056 Heusenstamm, Tel. (0 61 04) 33 13)

Creative Daten Systeme GmbH, Maria Eich Str. 11, 8032 Lochham, Tel. (0 89) 8 54 51 20

Data General (Alfred Neye Enatechnik GmbH, Schillerstr. 14, 2085 Quickborn/Hamburg, Tel. (0 41 06) 61 21)

Diehl Datensysteme GmbH dds, Veilhofstr. 6, 8500 Nürnberg, Tel. (09 11) 5 30 51

Dietz Computer Systeme, Solinger Str. 9, 4330 Mülheim/Ruhr, Tel. (02 08) 44 34–1

Digital Equipment GmbH, Wallensteinplatz 2, 8000 München 40, Tel. (0 89) 3 50 31

Exidy (Computershop GmbH, Unterortsr. 10, 6236 Eschborn, Tel. (0 61 96) 4 69 33)

Feltron Elektronik GmbH & Co. Vertriebs KG, Auf dem Schellerod 22, 5210 Troisdorf-Spich, Tel. (0 22 41) 4 10 04

FMG Corp. (INTERFACE AGE Verlagsges. mbH, Dahlienstr. 4, 8011 Vaterstetten, Tel. (0 81 06) 73 96)

Hewlett-Packard GmbH, Berner Str. 117, 6000 Frankfurt 56, Tel. (06 11) 5 00 41

Intel Semiconductor GmbH, Seidlstr. 27, 8000 München 2, Tel. (0 89) 5 38 91

ITT Standard Elektrik Lorenz AG, Hellmuth-Hirth-Str. 42, 7000 Stuttgart 40, Tel. (07 11) 8 21–1

Kontron Elektronik GmbH, Postfach 1251, 8057 Eching, Tel. (0 89) 3 19 01 (CSC GmbH, Straßbergerstr. 12, 8000 München 40)

Lucidata (ELTEC Elektronik GmbH, Neubrunnenstr. 10, 6500 Mainz, Tel. (0 61 31) 3 13 36)

Motorola GmbH, Münchner Str. 18, 8043 Unterföhring, Tel. (0 89) 9 24 81

MPDV, 6957 Ilztal-Dallau, Tel. (0 62 61) 1 44 48

OHIO Scientific (Balü Electronic, Burchardplatz 1, 2000 Hamburg 1, Tel. (0 40) 33 09 71 und Digitus Datentechnik, Am Schloßpark 12, 8035 Gauting, Tel. (0 89) 8 50 68 66)

OMSI (AC-Copy, Kurbrunnenstr. 30, 5100 Aachen, Tel. (02 41) 50 60 96 und PCS, Pfälzer-Wald-Str. 36, 8000 München 90, Tel. (0 89) 68 10 21)

RAMTEC (Logotec GmbH, Düsseldorf Str. 13, 8000 München 40, Tel. (0 89) 3 00 86 37 und WDV GmbH, Zeppelinstr. 11, 8046 Garching b. München, Tel. (0 89) 3 20 21 46)

rowi Rothdach und Wiesenhabart, Zusamstr. 8, 8900 Augsburg, Tel. (08 21) 7 30 48 (bitronic GmbH, Einsteinstr. 127, 8000 München 80, Tel. (0 89) 4 70 20 98)

SALOTA GmbH & Co. KG, Raabstr. 24, 8500 Nürnberg, Tel. (09 11) 3 21 77

Siemens AG, ZVW 104, Postfach 103, 8000 München 1

Spezial-Electronic KG, Hermann-Lingg-Str. 16, 8000 München 2, Tel. (0 89) 53 03 87

SWT (Digicomp AG, Werdstr. 38, CH-8004 Zürich, Tel. (01) 2 41 79 09)

TANDY Radio Shack, Luisenstr. 98, 4000 Düsseldorf, Tel. (02 11) 37 05 29

Texas Instruments Deutschland GmbH, Haggertystr. 1, 8050 Freising, Tel. (0 81 61) 8 01

Western Digital (Electronic 2000, Neumarkter Str. 75, 8000 München 80, Tel. (0 89) 43 40 61)

Zilog GmbH, Zugspitzstraße 4, 8011 Vaterstetten, Tel. (0 81 06) 40 35

Kurzberichte

Siemens kündigt PASCAL-Compiler für die Systeme 300 an

Nachdem für die Minicomputer der Systeme 300 bereits BASIC-, COBOL-, FORTRAN- und PEARL-Compiler existieren, wird Siemens im vierten Quartal dieses Jahres auch noch einen PASCAL-Compiler anbieten. Mit dieser umfassenden Software-Unterstützung will man den unterschiedlichen Anforderungen gerecht werden und auch komplexe Anwendungsfälle abdecken, wie z. B. die schnelle Reaktion auf externe Signale bei dynamischer Systemumgebung, die Steuerung paralleler Vorgänge oder die Koordinierung des Ein-/Ausgabe-Verkehrs bei umfangreichen Peripherie-Konfigurationen. Jede der angebotenen Compilersprachen bietet auf die speziellen Belange abgestimmte Eigenschaften, wobei PASCAL 300 die Technik der strukturierten Programmierung wirkungsvoll unterstützt und insbesondere Hilfestellung für die Ein-/Ausgabe und den Test gibt.

□ Hersteller: Siemens AG, ZVW 104, Postf. 103, 8000 München 1.

System-PASCAL deckt großen Anwendungsbereich ab

Als Software-Werkzeug zur Lösung komplexer DV-Probleme stellt die Firma Dietz System-PASCAL vor. Dieses Instrument ermöglicht es, umfangreiche Anwendungen in einem breitgefächerten Anwendungsbereich – von der Prozeßautomation bis zur Kommunikationstechnik – effizient und mit überschaubarem Aufwand zu realisieren. Nach Meinung der Firma stellt es einen weiteren Schritt zum universellen Einsatz der Sprache PASCAL dar. Ferner bietet Dietz für den technisch-wissenschaftlichen Bereich unter anderem das Programmpaket MINI-PLAN, ein Planungsinstrument für Konstruktion und Entwicklung, und ein interaktives CAD-System an. Letzteres stammt vom Dietz-Tochterunternehmen Technovision.

□ Hersteller: Dietz Computer Systeme, Heinrich Dietz, Solinger Str. 9, 4330 Mülheim, Tel. (02 08) 44 34-1.

Mikroprozessor PASCAL

Texas Instruments bietet ein Software-Paket an, das den Einsatz der höheren Programmiersprache PASCAL für Mikroprozessorentwicklungen auf dem Entwicklungssystem AMPL bzw. FS 990/4 ermöglicht. Mikroprozessor PASCAL (MPP) enthält neben einem Compiler ein ganzes Paket an Software-Unterstützung. So steht neben dem bildschirmorientierten Editor mit „Syntax-Check" der auf PASCAL-Sprachebene arbeitende Debugger zur Verfügung, der u. a. das Setzen von Breakpoints mit anschließender Ausgabe aller interessierenden Programmdaten ermöglicht. Weitere Debug-Routinen erlauben den Echtzeittest unter Zuhilfenahme aller Eigenschaften des AMPL. Der Run-Time-Support beinhaltet zwei Versionen: die eine zur interpretativen, die andere zur direkten Abarbeitung der erstellten Software im Anwendersystem. Damit hat der Anwender die Möglichkeit, sein PASCAL-Programm wahlweise speicherplatzoptimal oder zeitoptimal abzuarbeiten. Der Konfigurator erlaubt es, für die spezielle Anwendung nicht benötigte Routinen aus dem Run-Time-Support zu eliminieren und damit ein minimales Lademodul zusammenzustellen.

□ Vertrieb: Texas Instruments GmbH, Haggertystr. 1, 8050 Freising, Tel. (0 81 61) 80-1.

PASCAL für 8-Bit-Prozessor

Die Firma Motorola bietet zu ihren Entwicklungssystemen mit dem Mikroprozessor MC6809 einen PASCAL-Interpreter (M6809PASLI) an. Ein lauffähiges System benötigt eine Floppy-Disk mit dem Betriebssystem MDOS, ein Datensichtgerät und 56 KByte RAM. Das M6809-PASCAL hält sich erfreulicherweise eng an den Normentwurf der Sprache. Es bietet u. a. folgendes an Erweiterungen: nicht-dezimale Darstellung ganzer Zahlen, eine *exit*-Anweisung, eine *otherwise*-Klausel in der *case*-Anweisung, Operationen auf Zeichenketten. In ein übersetztes PASCAL-Programm lassen sich sowohl Assembler-Routinen als auch andere PASCAL-Unterprogramme einbinden. Die bei der ersten Auslieferung noch fehlenden Sprachelemente (Datentyp „real", gepackte Datenstrukturen und die formalen Prozedur- und Funktionsparameter) sowie Erweiterungen für dynamische Feldgrenzen, Echtzeit-Verarbeitung u. a. sind für die nächste Version angekündigt.

□ Vertrieb: Motorola GmbH, G.-B. Halbleiter, Pf. 1229, 8043 Unterföhring, Tel. (0 89) 92 48-1.

Pascal

OMSI PASCAL V1.2

Der OMSI-PASCAL-Compiler ist **der** Standard für alle PDP-11 und LSI-11 Systeme.

DESIGN — **Ein umfassendes Software-Werkzeug** — Das PASCAL V1.2-System ist das Werkzeug, das Sie für alle Phasen der Software-Entwicklung benötigen.

CODE & COMPILE — **V1.2 Pascal Compiler** — Pascal V1.2 bietet einen zuverlässigen Compiler, der sehr schnell lauffähigen Code erzeugt.

TEST — **V1.2 Symbolischer Debugger** — Der symbolische Debugger arbeitet auf Quell-Programm-Ebene. Er stellt Break Points und die Überwachung von variablen sowie andere attraktive Eigenschaften bereit.

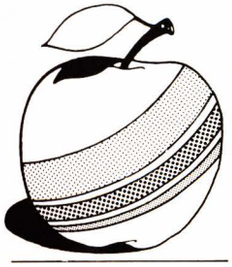
MEASURE — **V1.2 Profiler** — Der Profiler ermittelt die Anzahl der Durchläufe für die einzelnen Source-Statements und erlaubt dem Anwender damit die Optimierung seiner Programme.

OPERATE & MAINTAIN — **Übertragbar auf alle standard PDP-11 Betriebs-Systeme** — Pascal V1.2 Quellcode läßt sich auf RSX-11, IAS, RT11 und RSTS/E übertragen. Das erzeugte Programm läuft auf jeder PDP-11 – von der LSI-11 bis zur VAX.

Periphere Computer Systeme GmbH

PCS-GmbH
Pfalzer-Wald-Straße 36
D-8000 München 90
Telefon 0 89 / 68 10 21
Telex 5 23 271

PCS-GmbH · GS Düsseldorf
Borsigstraße 12
D-4030 Ratingen 1
Telefon 0 2102 / 47 00 09
Telex 08 5 315



PCS

Pascal

DAS SOFTWARE WERKZEUG

OMSI PASCAL V 1.2 für alle DEC PDP 11 - LSI 11 und VAX RSX-11, RT-11, IAS und RSTS/E.

- Vollständiges Standard PASCAL
- FORTRAN und MACRO Interface
- In line MACRO Code
- Schneller One-Pass-Compiler
- erzeugt Assembler Code

Debugger

- Arbeitet im PASCAL-Quellprogramm
- Setzen von mehreren Breakpoints möglich
- Überwachung von Variablen
- Programmlaufstatistik

OMSI PASCAL V 1.2 ist ein Produkt der Oregon Software, 600 mal installiert in 22 Ländern, Anwendung in Industrie, Lehre und Forschung und Verwaltung.

Weitere Informationen und Verkauf:

AC COPY

Ingenieurbüro für Datenverarbeitung GmbH
D-5100 Aachen
Kurbrunnenstraße 30
Telefon 0241/506096
Telex 0832368

PASCAL für DEC-Rechner

Die Firma AC Copy bietet einen PASCAL-Compiler für die Rechnerserien LSI-11, PDP-11 und VAX von DEC an. Diese PASCAL-Version mit der Bezeichnung Omsi wurde von der Oregon Minicomputer Software entwickelt. Seit 1975 wurde dieses PASCAL etwa 600mal installiert. Gegenüber dem Standard-PASCAL weist Omsi-PASCAL zahlreiche insbesondere für die Prozeßsteuerung nützliche Erweiterungen auf. Es ist beispielsweise möglich, absolute Speicheradressen anzusprechen oder im PASCAL-Programm Assemblerbefehle einzufügen. Omsi-PASCAL erzeugt Assemblercode, der dann in den Objektcode übersetzt wird. Die Programme benötigen zur Laufzeit außer dem normalen Betriebssystem keine weitere Unterstützung. Omsi-PASCAL ist für die Betriebssysteme RT-11, RSX, IAS und RSTS von DEC verfügbar. Besonders hilfreich ist der symbolische Debugger, der mit dem PASCAL-Quellprogramm arbeitet. Dadurch kann man Variable mit ihrem Namen ansprechen und untersuchen oder verändern, im Quellprogramm mehrere Stop-Punkte setzen, das Programm schrittweise ablaufen lassen usw. Der symbolische Debugger und weitere Programme sowie ein Listenformatierer gehören zum Standardlieferumfang.

□ Vertrieb: AC-Copy Ingenieurbüro für Datenverarbeitung GmbH, Kurbrunnenstr. 30, 5100 Aachen, Tel. (02 41) 50 60 96.

PASCAL-Compiler für PCS-SII

Für ihr Package-System PCS-SII, das auf dem LSI-11 basiert, hat die Firma PCS jetzt einen PASCAL-Compiler im Vertriebsprogramm. Dieser sowohl von RT-11 als auch von RSX-11 unterstützte Compiler eignet sich für den Einsatz in technischen und in kommerziellen Anwendungen. Da er nach dem Compilierungs-Vorgang zusätzlich eine Code-Optimierung durchführt, wird die Laufzeit auf ein Minimum herabgesetzt und der Speicherplatz optimal ausgenutzt.

□ Vertrieb: PCS GmbH, Periphere Computer Systeme, Pfälzerwaldstr. 36, 8000 München 90, Tel. (0 89) 68 10 21.

UCSD-PASCAL für SWT-6809

Das Mikrocomputersystem SWT-6809 von Southwest Technical kann mit dem Softwarepaket von Digicomp jetzt auch als PASCAL-Rechner betrieben werden. Voraussetzung dazu ist der Ausbau des Systems auf 56-K-RAM und der Anschluß einer 5- oder 8-Zoll-Floppy-Disk. Wegen der Größe des Softwarepakets ist eine 8-Zoll-Floppy vorzuziehen. Die im SWT-System lieferbare 16-MByte-Hard-Disk wird ebenfalls voll unterstützt. Das PASCAL-System ist die UCSD-Originalversion. Es umfaßt einen schnellen One-Pass-Compiler und einen P-Code-Interpreter. Das Linken von P-Code und 6809-Assemblerroutinen ist möglich. Ein PASCAL-Compiler, der direkt 6809-Maschinencode erzeugt, ist in Vorbereitung.

□ Vertrieb: Digicomp AG, Werdstr. 36, CH-8004 Zürich, Tel. (01) 2 41 79 09.

Mikrocomputer bietet komfortables PASCAL

Das von Bitronic vertriebene Mikrocomputersystem microwi 65 basiert auf der CPU 6502. Es bietet 14 Steckplätze für Europakarten. Beispielsweise läßt sich dort die 8-K-RAM-Karte einstecken, deren Adreßbereich sich frei wählen läßt. Das System besitzt eine V-24- bzw. 20-mA-Schnittstelle sowie eine Parallelschnittstelle mit 16 Bit für Druckeranschluß (incl. Software). Es werden 24 Zeilen mit jeweils 40 oder 80 Zeichen dargestellt, wobei Groß- und Kleinschreibung sowie 128 Grafikzeichen möglich sind. Die Minidiskette speichert 91 KByte. Das System ist mit einem Doppellaufwerk erweiterbar. An Software bietet das System einen Editor, Assembler, UCSD-PASCAL und BASIC. Mit PASCAL ist das System ab 12 995,– DM erhältlich.

□ Vertrieb: Bitronic GmbH, Einsteinstr. 127, 8000 München 80, Tel. (0 89) 4 70 20 98.

Softwarepaket PASCAL-80

Für den Einsatz in dem Intellec-Mikrocomputer-Entwicklungssystem bietet Intel ein Softwarepaket PASCAL-80 an. Es besteht aus einem Compiler, Pseudocode-Interpreter und einem Demonstrationsprogramm auf Diskette, einem Anwender-Handbuch und der zweiten Auflage des Buches „PASCAL user manual and report" von Jensen/Wirth. PASCAL-80 läuft auf dem Intellec-Serie-II und MDS-800 unter dem ISIS-II-Betriebssystem.

Gegenüber dem akademisch orientierten Standard-PASCAL ist diese Version für kommerzielle und industrielle Anwendungen erweitert. So wurden drei neue Datenarten eingeführt – String, Untyped Files und Interactive Files – sowie 28 Vorgänge und Funktionen. Dadurch müssen Variable für häufig benutzte Vorgänge und Funktionen nicht einzeln definiert werden. Zusätzlich besteht die Möglichkeit der Überwachung der Programmausführung und der Fehlerdiagnose. PASCAL-80 wurde exklusiv für Intel von der Queue Computer Corporation entwickelt und ist unter der Produktbezeichnung MDS-381 zu beziehen.

□ Vertrieb: Intel Semiconductor GmbH, Seidlstr. 27, 8000 München 2, Tel. (0 89) 53 89-1.

Auch für TRS-80 gibt es PASCAL

Ein breites Software-Programm für TRS-80-Anwender vertreibt in Europa die Interface-Age Verlags GmbH. Es handelt sich unter anderem um FORTRAN-80, UCSD-PASCAL, TRS/COBOL-80, Text-Editor, Screen-Editor usw. Die Manuals dazu sind auch einzeln erhältlich. Ein Katalog mit näheren Beschreibungen ist verfügbar.

□ Vertrieb: Interface Age Verlags GmbH, Dahlienstr. 4, 8011 Vaterstetten, Tel. (0 81 06) 73 96.

Fachbücher

An Introduction to Programming and Problem Solving with PASCAL

Von G. Michael Schneider, Steven W. Weingart und David M. Perlman. 394 Seiten, 115 Bilder. INTERFACE AGE, Vaterstetten. ISBN 0-471-04431-8, DM 39,–.

Obwohl bereits einige Lehrbücher für das Programmieren mit PASCAL in deutscher Sprache vorliegen, gibt es gute Gründe, dieses amerikanische Lehrbuch zu empfehlen. Es bietet eine Einführung in das Programmieren in all seinen Aspekten: Problem-Definition, Suchen der Lösung, Auswahl des Algorithmus, Codieren des Programms, Austesten und Fehlersuche, Dokumentation und Wartung. Die Zielsetzung der Verfasser, einen guten Programmierstil an Hand von PASCAL einzuüben, ist erreicht worden. Dies geschieht durch eine ausführliche Erläuterung der grundlegenden Konzepte von PASCAL, durch die Besprechung häufig vorkommender Programmierfehler und durch Hinweise auf das Schreiben von verstehbaren Programmen.

Hervorgegangen aus einem einführenden Programmierkurs für Studenten aller Fakultäten, erfordert das Lehrbuch keine speziellen Vorkenntnisse. Sieht man von der kleinen Hürde des Englischen ab, so ist es besonders für Anfänger im Programmieren – auch im Selbststudium – geeignet. Für den Fortgeschrittenen bietet es eine vollständige Beschreibung von PASCAL und viele nützliche Hinweise zum Programmieren selbst. Abschließend sei bemerkt, daß das beschriebene PASCAL mit dem Standard von Jensen/Wirth übereinstimmt. *EE*

PASCAL für Anfänger

Programmieren für Anfänger. Dritte Auflage. Von Helmut Schauer, 175 Seiten, zahlreiche Programmlisten. R. Oldenbourg Verlag, München. ISBN 3-486-20233-2, DM 21,80.

Diese Einführung in die Programmiertechnik soll einen Anfänger mit dem Problemkreis des Softwareentwicklers vertraut machen. Das logische und algorithmische Denkvermögen des Lesers wird geschult, so daß er in der Lage ist, selbst Programme zu schreiben. Der inhaltliche Aufbau des Buches ist sehr induktiv, es wird von konkreten Beispielen ausgehend auf allgemeine Sachverhalte geschlossen. Die Programmbeispiele sind frei von unnötigem Ballast, was ein besonderer Vorzug der Programmiersprache PASCAL ist. Die Prinzipien jedes Kapitels werden an zahlreichen Beispielen illustriert. Für jene Leser, die praktische Programmiererfahrung gewinnen wollen, wird eine Sammlung zusätzlicher Übungsaufgaben mit den dazugehörigen Lösungen in einem eigenen Band PASCAL-Übungen zusammengestellt. Der letzte Abschnitt, mit der Überschrift Das Buch enthält eine Zusammenfassung der wichtigsten syntaktischen Regeln von PASCAL. Das erleichtert das Nachschlagen in Zweifelsfällen sowie das Erlernen dieser Regeln. *May*

Elektronik

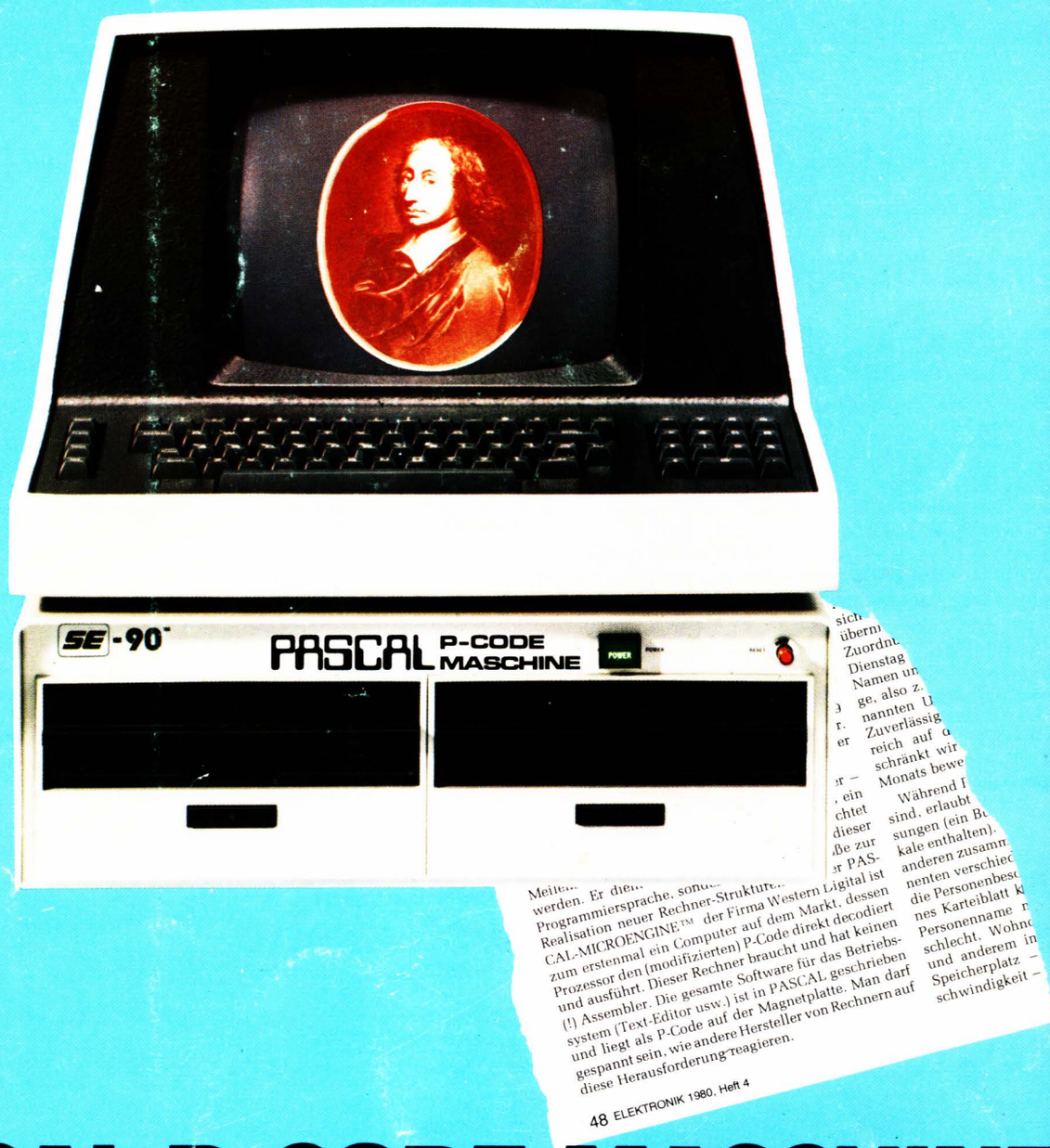
...die Sicherheit besser informiert zu sein

Die Elektronik revolutioniert beinahe jede Branche und jedes Arbeitsgebiet. Daraus ergibt sich ein gewaltiger Informationsfluß, der nur mit viel Sachkenntnis und Umsicht bewältigt werden kann. Diese Auswahl der relevanten Informationen bietet Ihnen die Fachzeitschrift ELEKTRONIK, damit Sie immer zum richtigen Zeitpunkt über alles Wesentliche informiert sind. Eigene Korrespondenten in den USA und Japan garantieren eine stets aktuelle Berichterstattung auch von diesen Märkten. Exklusive Copyright-Abkommen mit führenden amerikanischen Fachzeitschriften ermöglichen die Übernahme von Aufsätzen über neueste Technologien in deutscher Sprache.
Die ELEKTRONIK berichtet über: Meß-, Steuer- und Regelungstechnik, Analog- und Digitaltechnik sowie Datentechnik, mit Anwendungen in Forschung und Industrie, Maschinen- und Gerätebau, Verkehrswesen und Medizin. Industrielle Prozeßautomatisierung, Fertigungsmethoden für die Elektronik-Industrie. Bauelemente einschließlich neuer Technologien und Anwendungen, Schaltungstechnik und -praxis, Nachrichtentechnik, Optoelektronik. Mikroprozessor und Mikrocomputer werden in der ELEKTRONIK besonders intensiv betreut. Auch die Programmierung (software) wird eingehend behandelt. ELEKTRONIK-Beiträge sind praxisnah geschrieben und damit bei den täglich anfallenden Problemen direkt verwertbar. Sie vermitteln die oft teuer bezahlte Erfahrung anerkannter Experten und bringen Grundlagen des Ingenieurwissens in Form der *Elektronik-Arbeitsblätter*, marktgerechte Neuheitenberichte im *Elektronik-Markt* und aktuelle Branchen-Nachrichten im *Elektronik-Express.*

Zwei *Lexikon-Karten* und zwei *Normen-Karten* pro Heft bereichern laufend den technischen Wortschatz bzw. ermöglichen den Aufbau einer Normenkartei. Der Rubrik *ELEKTRONIK-Notizen* kann man mit einem Blick das Neueste aus Forschung und Anwendung der Elektronik entnehmen. Eine umfassende internationale Literaturschau vermittelt in jedem Heft den unerläßlichen „Blick über den Zaun".

Bitte bestellen Sie mit nebenstehender Bestellkarte

Die ELEKTRONIK erscheint 25mal im Jahr alle 14 Tage am Donnerstag.
Das Einzelheft kostet DM 4.80,
das Jahresabonnement DM 94.– im Inland,
DM 104.– im Ausland
(ab 1. 1. 81 DM 104.40 bzw. DM 111.60).



PASCAL P-CODE MASCHINE

Sie und wir haben lange gewartet. Doch es hat sich gelohnt! Denn jetzt bekommen Sie die zweite Generation: mit integrierten Floppy-Laufwerken, einfache oder doppelte Schreibdichte.

- PROZESSOR: PASCAL-P-CODE-CHIP-SATZ
- SPEICHER: 64 kByte RAM
- SCHNITTSTELLEN: 2 serielle V24-Schnittstellen (synchron/asynchron)
  1 paralleles E/A-TOR (8 BIT)
- SOFTWARE: UCSD-Betriebssystem III.0 mit den Moduln:
  - Editor
  - File-Handler
  - PASCAL-Compiler
  - Linker
  - Library
  - und mit diversen Utility-Programmen.

Jetzt können Sie auf den PASCAL-Zug aufspringen! Sofort!

Mit der SE-90/S für DM 11.290,– (mit einseitigen Floppy-Disk-Drives)
oder der SE-90/D für DM 12.500,– (mit doppelseitigen Floppy-Disk-Drives)

SE Spezial-Electronic KG

3062 Bückebürg/Hannover, Kreuzbreite 14, Tel. 05722 1011-15, Telex 9 71 624 spez d
8000 München 2, Hermann-Lingg-Straße 16, Tel. 089 53 03 87, Telex 52 12 176 spez d
Außenbüro: 7090 Ellwangen, Magdeburger Str. 15, Tel. 0 79 61-40 47, Telex 74 712